Performance of Computer Systems

Edited by
M. ARATO
A. BUTRIMENKO
E. GELENBE

NORTH-HOLLAND

PERFORMANCE OF COMPUTER SYSTEMS

Sponsored by

IRIA
IFIP Working Group 7.3
SZÁMKI
IIASA
Technische Universität Wien
GMD
EURATOM

PERFORMANCE OF COMPUTER SYSTEMS

Proceedings of the 4th International Symposium on
Modelling and Performance Evaluation of Computer Systems,
Vienna, Austria, February 6-8, 1979

Organized by the
International Institute for Applied Systems Analysis

Edited by

M. ARATÓ
SZÁMKI, Budapest, Hungary

A. BUTRIMENKO
IIASA, Laxenburg, Austria

E. GELENBE
IRIA-LABORIA, Rocquencourt, France

1979

NORTH-HOLLAND PUBLISHING COMPANY
AMSTERDAM • NEW YORK • OXFORD

ISBN: 0 444 85332 4

Publishers:
NORTH-HOLLAND PUBLISHING COMPANY
AMSTERDAM · NEW YORK · OXFORD

Sole distributors for the U.S.A. and Canada:
ELSEVIER NORTH-HOLLAND, INC.
52 VANDERBILT AVENUE,
NEW YORK, N.Y. 10017

Library of Congress Cataloging in Publication Data

International Symposium on Modelling and Performance Evaluation of Computer Systems, 4th, Vienna, 1979.
Performance of computer systems.

Includes index.
1. Electronic digital computers--Evaluation--Congresses. I. Arató, Mátyás, 1931- II. Butrimenko, Alexandre. III. Gelenbe, E., 1945- IV. International Institute for Applied Systems Analysis.
V. Title.
QA76.9.E94I58 1979 001.6'44'04 79-14176
ISBN 0-444-85332-4

Printed in The Netherlands

PREFACE

The Fourth International Symposium on Modelling and Performance Evaluation of Computer Systems organized in Europe by the IFIP Working Group 7.3 was held in Vienna under the auspices of the Technical University and the International Institute for Applied Systems Analysis. This series of conferences has been placed under the auspices, successively, of the following major European research organizations: IRIA, France (1974), EURATOM (1976) and GMD, Federal Republic of Germany (1977). Since 1978, IFIP WG 7.3 has decided to hold these meetings alternately in North America and in Europe approximately every eighteen months.

These proceedings are a collection of contributions to computer system performance, selected by the usual refereeing process from papers submitted to the symposium, as well as a few invited papers representing significant novel contributions made during the last year. They represent the thrust and vitality of the subject as well as its capacity to identify important basic problems and major application areas. The main methodological problems appear in the underlying queueing theoretic aspects, in the deterministic analysis of waiting time phenomena, in workload characterization and representation, in the algorithmic aspects of model processing, and in the analysis of measurement data. Major areas for applications are computer architectures, data bases, computer networks, and capacity planning.

The international importance of the area of computer system performance was well reflected at the symposium by the presence of participants from nineteen countries. The mixture of participants was also evident in the institutions which they represented: 35% from universities, 25% from governmental research organizations, but also 30% from industry and 10% from non-research government bodies. This proves that the area is reaching a stage of maturity where it can contribute directly to progress in practical problems.

The Editors.

LIST OF CONTENTS

METHODOLOGY

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

A Systematical Approach to the Performance Modelling of Computer Systems

M. G. Kienzle
Thomas J. Watson Research Center
Yorktown Heights, New York, USA

K. C. Sevcik
Computer Systems Research Group,
University of Toronto, Toronto

Abstract

This paper proposes a modeling procedure that relates the different aspects of modeling: model design, system measurement, parameter estimation, model analysis, model validation, performance prediction. The modeling procedure is organized in several steps that isolate the different abstraction steps to provide a better understanding of the modeling process. The proposed procedure is applied to a case study.

1. INTRODUCTION

With the increasing complexity and cost of computer systems, the need for efficient management tools for them is growing as well. In particular, tools for performance analysis and prediction are needed for installation planning, system development, and system tuning.

Since the behaviour of actual computer systems is too complex to be comprehended or predicted merely by inspection, models are used for their analysis. These models are abstractions of the systems, representing only those aspects of the systems that are relevant for the particular analyses. The models must not only represent the important system components, but also characterize quantitatively the relationships and interactions of these components.

One class of models that is suitable for these analyses and that has received much attention recently are queueing network models. They represent the components of a computer system, such as the CPU, I/O channels and devices, as servers with queues. These servers are connected by paths to form a network. The programs being executed in the system are represented in the model by tokens that circulate through the network.

There are three major methods by which queueing network models can be analyzed. In *analytic queueing network models,* the system components and their interactions are represented mathematically, and the models are evaluated using mathematical algorithms.

In *simulation models,* the operation of the system is simulated by a computer program. *Hybrid models* combine these two approaches.

Quite a number of case studies using queueing network models have appeared, e. g., [ROSE75, GIAM76a, BARD77, KGT77, SU78, BUZE78]. However, most of these papers have concentrated on the mathematical aspects of the models, and perhaps some practical problem. They all show the absence of a general modeling methodology, a modeling approach that would relate all the aspects of modeling: model design, system measurement, parameter estimation, model analysis, model validation, performance prediction.

In this paper, we propose such a modeling methodology. We develop a modeling process that includes all the above aspects of modeling. By isolating these aspects and by making all modeling decisions apparent, a better understanding of the modeling process is attained. The usefulness of this approach is demonstrated by applying it to an analytic queueing network model. It could also be applied to simulation or hybrid models.

In section 2, the series of abstraction steps involved in modeling and an overall modeling procedure are outlined briefly. They are more extensively described in the following sections using a modeling case study of an actual computer system as an example. Sections 3 to 6 are each divided into two parts: in the first part, concepts are developed, and in the second part, these concepts are applied to the case study. In section 3, the preliminary phase of designing the models is described. In section 4, the model used for the measurement experiments is developed. In section 5, a general model for parameter estimation is introduced. In section 6, the computational model used for the the mathematical analysis is defined, and some results are shown. Finally, in section 7, some conclusions are drawn.

2. MODELING PROCESS

2.1 Abstraction Steps

The process of modeling can be viewed as a series of abstraction steps leading from the actual computer system to the computational model that yields the desired performance measures. In the proposed modeling procedure, these steps are isolated, and intermediate models are defined in order to form a systematic approach to the modeling process. This stepwise approach reveals the decisions made in applying the abstractions and associated underlying assumptions.

There are three major models involved in the abstraction process from the actual computer system to the final computational model. These models and the abstraction steps are shown in figure 1.

The *measurement model* provides the basis for a measurement experiment. The variables it includes are measureable system variables, that is, variables that can be recorded by

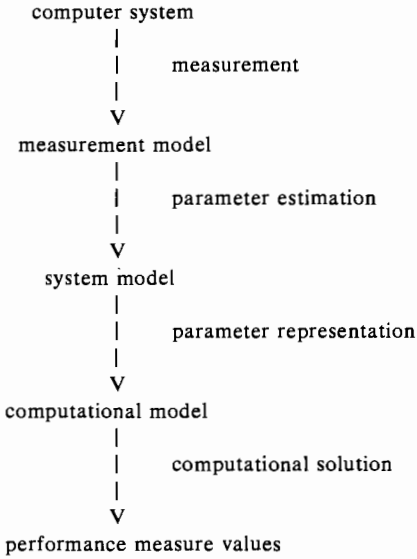


Figure 1. Abstraction steps.

hardware or software monitors. For each measurement experiment, a specific measurement model should be used.

Parameter estimation [DB78] is the step from the measurement data to the logical system model. In the parameter estimation, data from several experiments as well as general information on the system such as the service rates of devices or access times to memory are used to build the system model.

In the *system model,* all essential parts and functions of the system are represented from a logical point of view. They are looked upon as service facilities for user programs. The system model is introduced to describe separately the major components that influence a system's performance. It should give insight into the operation of the computer system and aid the interpretation of the model results. Most computational models are too restricted to serve these purposes.

The *parameter representation* is the mapping of the detailed functional system model parameters into the usually less detailed input parameters of the computational queueing network model as required by the solution algorithm.

The parameter requirements of the *computational queueing network model* [DB78] are determined by its type and by the solution algorithm selected. The parameters do not necessarily reflect the operation of the system explicitly.

In the *computational solution,* the desired performance measure values are calculated from the parameters of the queueing network models by the solution algorithm chosen.

The *performance measure values,* finally, are the goal of this abstraction process. The performance measures of interest are determined by the aims of the study.

2.2 Overall Modeling Procedure

Queueing network models are mainly used to predict the performance of a system under changes in its configuration, its software, or its workload. When modeling for performance prediction, three phases in the modeling procedure can be distinguished.

In the *design phase,* the three models used in the abstraction process are defined. Based on the objective of the modeling study, the abstraction steps are planned, considering in particular the compatibility of the three models involved.

In the *validation phase,* the abstraction steps are applied to the computer system. The results of the computational model must be validated against observed performance. If the agreement is poor, the decisions taken in the design phase must be reexamined.

In the *prediction phase,* the system changes that are to be investigated are applied to the system model. New input parameters for the computational model are calculated, and the solution of the computational model yields the predicted performance measure values for the changed system.

3. DESIGN PHASE

3.1 Concepts

At the beginning of a modeling study, the goals of the study and the desired performance measures must be defined as precisely as possible. It is necessary to have a comprehensive description of the system including the following topics: the system configuration, the operating system logic, the workload of the system, the monitoring facilities, and the operating aspects of the system. Based on this information, the three models can be designed in a way that each model supplies all the information required in the next abstraction step. From the desired performance measures, a computational model capable of producing these measures is determined. From its input parameter requirements, and based on the objective of the modeling experiment, the system model is defined. Finally, one or several measurement models are defined to supply the data for the system model. Care must be taken that the level of detail in each of the models involved is consistent. If, for example, the computational model requires not only the means but also the second moments of the service time distributions, the measurement models become more complex, and the measurement experiments may become prohibi-

tively expensive. As a consequence, for each system model the minimal input requirements should be stated to make the modeling experiment as simple as possible. For a survey of the types of analytic queueing network models and the parameter specifications for them see [KIEN77].

In this design phase, the model types, the parameter requirements, and the levels of detail are determined. The models cannot be defined in detail, and the design is not finalized since many decisions are dependent on quantitative aspects determined in the validation phase.

3.2 Case Study

In order to gain experience with the modeling procedure, a case study was carried out. The system oriented performance measures under consideration were the utilizations of the major hardware components of the system along with their mean queue lengths, and the job throughput, all by job class. As user oriented performance measures, the mean response time for interactive jobs and the mean residence time for batch jobs (excluding the time in the spooling system) will be computed by the model.

The system to be modeled is an IBM S/370-165 II of the University of Toronto Computer Centre (UTCC) running under the OS/VS2(MVS) operating system. It supplies batch and time sharing services (JES2, TSO). The configuration of the system is shown in figure 2.

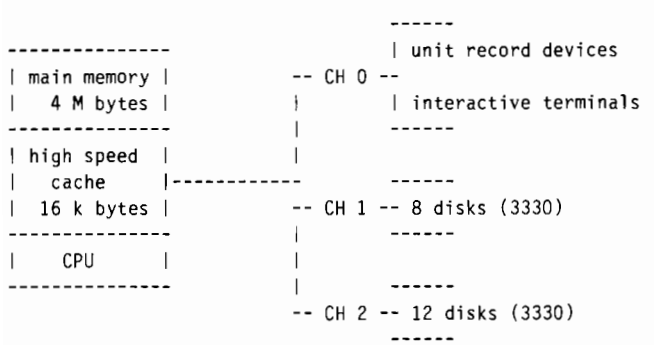


Figure 2. System configuration.

The System Resources Manager (SRM) of the operating system tries to keep the overall load balanced by initiating or swapping jobs as required. At the same time, it attempts to distribute the resources of the system among the jobs according to a predetermined pattern specified by the installation. Three time intervals defined in the operating system are important from a modeling point of view. The *residence time* of a task is the time it

belongs to the active mulitprogramming mix. The *elapsed time* of a task is the time from its initiation until its termination. The *time-in-system* of a task is the time from when it is read into the system until its termination. The memory management is based on a global, variable partition strategy, which means that the partition sizes and the number of active partitions are variable. The workload of the system consists mainly of three types of jobs:

1) compute oriented batch jobs (General Purpose Job Stream, *GPJS)*

2) small, students' jobs (High Speed Job Stream, *HSJS)*

3) interactive TSO jobs *(TSO)*

The system is measured under a benchmark that has one jobstream for each type of job. The TSO jobstream is generated by a simulator that uses a TSO command script. Several monitoring tools were activated during the execution of the benchmark. The software monitor RMF [IBM76b] is a sampling monitor that records the usage of the system's resources. The software monitor SMF [IBM77] reports the service requests by the jobs to the system. The hardware monitor TORMON [CM73] records data on the CPU and the channels to verify and to extend the data collected by RMF.

The computational model used is a hierarchical model. On the level of the dispatcher, a multiple class queueing network model in local balance is used to represent the CPU, the channels, and the disks. In this model, only the resident jobs will be represented. The three job classes GPJS, HSJS, and TSO are distinguished in the model. For the TSO interactions, an additional birth-death model is built on top of the queueing network model. The server of this model represents the entire computer system, and the service rates for TSO interactions are determined by the queueing network model. This way, the queueing of the interactive jobs before entering the set of resident tasks can be modeled.

Local balance models [BCMP75] do not require any information on the dynamic behaviour such as the burst lengths or the branching probabilities of the jobs in the system [KR75, GIAM76b]. Thus, accumulated values like total CPU time per job class and total number of I/O operations to a device are sufficient to determine the values of the input parameters to the computational models.

For this case study, no event trace monitoring is needed, since the system model does not require information on the dynamic behaviour of tasks. Only accumulative data on the job's service requests and on the usage of the system's resources are required. These data can be obtained by the monitors listed above.

4. MEASUREMENT MODEL

4.1 Concepts

The measurement model provides a framework for the actual measurement experiments. It consists of a set of measureable system state variables. The domains of these variables must be precisely defined. Consider, for example, a state variable called CPU state. It could have different domains such as (idle, busy), or (idle, system state, problem state), or (idle, system state, user-task-1, user-task-2, ..., user-task-n). A value change in a state variable is a state change event. Based on such a state model of the system, the quantities recorded by the monitors can be defined precisely. They may be defined, for example, as numbers of transitions between two states, numbers of times a state was entered, or total time a state was occupied. Event trace monitors record the state transition events and their state contexts. Sampling monitors observe the states of the system at time intervals independent of the system's operation. It may not always be necessary to define the state model explicitly in order to define the measurement requirements. However, care must be taken to insure that all the recorded variables and events are specified.

Often, the system variables that are easily recordable are not the variables that are required by the system model. In this case the recorded data must be aggregated, correlated, or deaggregated to obtain the desired parameters of the system model. If no directly related variables are measurable for some system model paramters, some indirectly related variables must be measured, so that the system model parameters can be estimated.

4.2 Case Study

The quantities measured for the case study and the values resulting from a measurement experiment on a benchmark are shown in table 1. Also, the monitors by which the data are collected are indicated.

5. SYSTEM MODEL

5.1 Concepts

The system model represents the computer system from a logical point of view. It is limited neither by the available monitors (as are the measurement models), nor by the solution algorithm (as is the computational model). It can be based on data from several measurement experiments, and it can be used for different types of computational models.

Table 1. Measurement data.

General Parameters

measurement interval 2108 secs (RMF)

	GPJS	TSO	HSJS	Source
jobs completed	56	950	308	(SMF)
total resident time (secs).	4564	4252	1268	(SMF)
job stream elapsed time (secs)	2108	1734	1268	(SMF)
average number of ready tasks	4.84	8.18	1.00	(RMF)
accounted CPU time under TCBs (secs)	784.9	214.4	306.2	(SMF)

other CPU Parameters:

busy time	1728 secs	(TORMON)
problem state time	774 secs	(TORMON)
system state time	954 secs	(TORMON)
total instructions	3.58*10E9	(TORMON)
high speed cache accesses ..	5.04*10E9	(TORMON)
high speed cache hit ratio .	.9555	(TORMON)
utilization	.8143	(RMF)

Memory and Load Management Parameters:

	GPJS	TSO	HSJS	source
swaps	93	1361	1	(SMF)
pages transferred for swaps	5184	37666	61	(SMF)
pages transferred for address space page faults	2390	10279	825	(SMF)
pages transferred for common area page faults	1781	90	2	(SMF)
VIO EXCPs	12836	150	83	(SMF)

VIO SIOs (total): 5869 (RMF)

Table 1 contd.

I/O Parameters:

address	SIOs (RMF)	EXCPs (SMF)			utilization (RMF)
		GPJS	TSO	HSJS	
channel 1	47927	8261	-	-	.2686*
channel 2	72387	17649	1324	1666	.3461*
disk 11	10236	-	-	-	.1586
disk 12	2355	-	-	-	.0482
disk 13	702	-	-	-	.0208
disk 14	12594	8261	-	-	.2285
disk 15	17982	-	-	-	.3503
disk 16	3634	-	-	-	.0935
disk 17	424	-	-	-	.0113
disk 21	2285	-	-	-	.0444
disk 22	749	-	-	-	.0104
disk 23	434	-	-	-	.0094
disk 24	8689	-	-	-	.2266
disk 25	9682	12685	-	-	.1709
disk 26	3447	-	-	-	.0463
disk 27	5846	86	-	-	.0973
disk 28	31053	-	-	907	.4117
disk 29	10098	4874	1325	759	.1161

Spooling Parameters:

	GPJS	TSO	HSJS	source
cards read	15352	925	32545	(SMF)
lines printed	184517	2300	53780	(SMF)

* values obtained by TORMON

The system model covers all the details represented explicitly in the intended computational model, as well details represented only implicitly in the computational model. For example, I/O channels and devices should be represented separately in the system model, but in the computational model they are often represented as aggregate servers. Beyond the details required for the computational model, the system model should at least capture all the details that are necessary to represent explicitely the system features that are being analyzed. In most cases, the system model is not a solvable queueing network model since it may contain aspects of the real system such as priority service disciplines that cannot currently be represented in an efficiently solvable computational model.

The system model consists of several parts, each representing one aspect of the computer system. The *program behaviour model* characterizes the behaviour and the service requirements of the user programs. The *interference pattern* captures all interference among user tasks and the operating system. The *multiprogramming mix* specifies the job mix under which the model is to be evaluated. The *workload model* specifies the workload for the modeled hardware resources. It is derived from the program behaviour model and the interference pattern. The *resource attributes* describe the properties of the modeled service facilities and the manner in which they process the service requests of the user tasks. The structure of the system model is shown in figure 3.

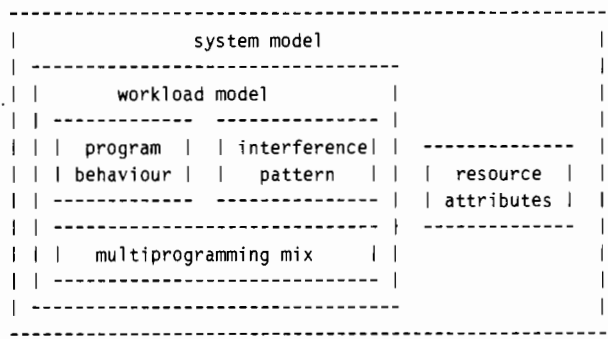


Figure 3. System model.

Program Behaviour Model

The program behaviour model captures the total service demand user tasks place on each of the system's resources. In addition, it captures the dynamic behaviour of user programs in the computer by specifying the patterns of their service demand on the system's resources like CPU burst time distributions, the probabilities of transitions of programs from one server to another, etc. To achieve hardware independence, the service demand is specified in numbers of operations rather than in service times or total times spent at devices.

In order to obtain the desired isolation of the user programs from the operating system, only service requests made directly by the user programs should be included. All interference from the operating system or from other user tasks, such as swaps, page faults, or I/O interrupts, should be omitted from the program behaviour model. Where this is not possible, a certain isolation can be achieved by specifying the operating system activities as functions of the user program behaviour, as is done in the interference pattern.

In local balance computational models, the program behaviour model can ignore all dynamic aspects of a task's behaviour. It need only contain the mean number of service requests per task and device, broken down by class. This information can be displayed in the system model using a *service request matrix* that contains an entry for each resource and job class pair. It has as its elements the average number of service requests to a certain device by a task of a certain class.

Interference Pattern

User programs do not see the hardware directly. They see an abstract machine that consists of the operating system and the hardware. Queueing network models, however, usually represent only the hardware components of a system. The gap between the service requests of the user programs and the hardware is bridged by the interference pattern. The interference among user tasks and the operating system has two major aspects. The *dynamic interference* reflects the impact on the dynamic behaviour of the tasks. The *static interference* maps the number of logical service requests into the number of physical operations, and it distributes the operating system overhead among the user tasks.

In order to specify the static interference pattern in detail, the amount of operating system activity related to the various system services must be determined. Cause and effect relationships must be established. It is most important to find the "correct" independent variables of the functions. The mapping of user EXCPs into SIOs, for example, may be a constant ratio, whereas CPU overhead is probably dependent on the multiprogramming mix, the number of page faults, and similar parameters. These dependencies should be established according to the operating system logic, and, where this is not possible, by making educated assumptions about their nature. They must be quantified by measurements taken under various load situations of the system, and they should be verified by modeling experiments using measurement data different from those used for establishing the interference functions.

Another important aspect is how the overhead is represented in the queueing network model. In this case study, the overhead resource usage is distributed among the jobs that cause it. A different approach is to represent it as a separate task, or a separate class of tasks [KGT77].

Multiprogramming Mix

The *multiprogramming mix* is a vector that indicates, for each class, the number of active jobs. For a single class, it is called the *multiprogramming level.* Because the multiprogramming level of various classes may change over time, the average multiprogramming level may involve nonintegral quantities.

Workload Model

From the program behaviour model and the interference pattern the *workload matrix* is established. This is a matrix similar to the service request matrix. However, instead of service requests per task and resource for each class it contains the total workload a task of a certain class places on a hardware device, broken down by class. The workload model is hardware independent in that the service demand placed on hardware components is given in numbers of physical operations.

Resource Attributes

The resource attributes capture the resource related aspects of the system model. They contain the service disciplines of all modeled resources. For hardware components, they also cover the service rates, specified in numbers of operations per unit time. For software resources a mapping into the number of operations on hardware devices is required. Load dependent service rates of service facilities like disk drives or entire I/O subsystems are described by a set of service rate values, one for each possible loading situation.

5.2 Case Study

For the two computational models used, the queueing network model and the birth-death model, two separate system models are defined.

5.2.1 System Model of the Multiclass Queueing Network Model

Program Behaviour Model

The service request matrix is calculated from the accounted service requests. Because the service demands of jobs belonging to the same class are, at least to a certain degree, homogeneous, these "average" jobs can be viewed as prototype jobs for their class, and their performance measures also have meaning to the users. The service requests of the three average jobs are listed in table 2.

Interference Pattern

Because the chosen computational model is based on an assumption of local balance, no dynamic interference need be considered. If no changes to the operating system are to be modeled, it is not crucial to distinguish between the operating system and user program usage of the physical resources accurately as long as the operating system overhead is attributed to the class that causes it. However, it is essential that the unaccounted resource usage be credited to the classes in the correct proportions. Due to

Table 2. Program behaviour for the multiple class queueing network model (request matrix).

	GPJS	TSO	HSJS
total number of tasks	56	950	308
CPU *	29.06	.468	1.186
channel 1	147.51	-	-
channel 2	315.16	1.394	5.409
disk 14	146.51	-	-
disk 25	222.52	-	-
disk 27	1.57	-	-
disk 28	-	-	2.945
disk 29	87.04	1.395	2.464
pages transferred for VIO	229.21	0.158	0.269
cards read	274.0	1.0	106.0
lines printed ..	3735.0	92.0	174.0

* in millions of instructions

limited time available for monitoring in this case study, no special measurements could be made to establish cause and effect relationships, and to distribute the unaccounted resource usage depending on measureable system variables. An accurate breakdown of usage by job class is not available for any physical resource. Instead, estimates of the actual breakdown are made based on related data. For instance, the start I/Os (SIOs) to the spooling disk can be distributed among the classes according to the accounted numbers of cards read and lines printed, and the SIOs to the page data sets can be distributed according to the numbers of pages transferred. For cases where no related data can be measured the overhead is distributed according to the following ratios:

GPJS: .276
TSO: .600
HSJS: .124

These numbers were determined by giving equal consideration to the number of job step intiations per class and the total number of I/O operations.

Multiprogramming Mix

For the TSO class, we do not need to specify a particular multiprogramming level. Several multiprogramming levels are evaluated, and the performance measures are weighted by the results of the birth-death model for the TSO transactions. For GPJS, the average multiprogramming level is calculated from the sum of the resident times:

$$MPL = \frac{\sum \text{resident times}}{\text{measurement interval}}$$

The HSJS can have a multiprogramming level of at most one. So its average multiprogramming level can be calculated as the class elapsed time divided by the measurement interval. The resulting average multiprogramming mix is:

GPJS: 2.1
HSJS: 0.6

(For TSO, we will investigate multiprogramming levels between 0 and 3.) Using the mean multiprogramming levels ignores the large variation in the multiprogramming levels. It also ignores the heavy interaction between the multiprogramming levels due to the load balancing efforts of the SRM and the different job class elapsed times. However, considering the information available, this is the best approximation we can make.

Workload Model

The workload model must be specified in such a way that the parameters of the solution algorithm can be derived easily. Algorithms assuming local balance have as their major input parameters relative utilizations. The relative utilizations are guaranteed to be in the same ratio as the total busy times of the servers. There are several equivalent ways of specifying a consistent set of relative utilizations using data of different levels of detail.

a) total load during T: $X(i,r) = L(i,r) / C(i)$

b) load per partition: $X(i,r) = L(i,r) / (MPL(r) * C(i))$

c) load per job: $X(i,r) = L(i,r) / (jobs(r) * C(i))$

d) load per cycle: $X(i,r) = L(i,r) /(cycles(r) * C(i))$

T: measurement period
X(i,r): relative utilization of server i due to tasks of class r
L(i,r): number of operations executed by server i for tasks of class r
C(i): service rate of server i, assumed to class independent
MPL(r): mean multiprogramming level of class r
Jobs(r): number of class r jobs completed
Cycles(r): total number of cycles of class r jobs through the network

It can be shown easily that these definitions are mathematically equivalent [KR75,GIAM77], and that they require the same measurement data. Since the relative utilization will be determined from the program behaviour model, we chose the job oriented definition, c) that directly gives the elements of the workload matrix. The workload matrix is shown in table 3.

Table 3. Model workload (workload matrix).

address	GPJS	TSO	HSJS
CPU *	33.33	1.02	2.41
channel 1	488.7	16.4	16.3
channel 2	404.0	48.1	13.0
disk 11	126.0	0.12	9.91
disk 12	11.6	1.48	0.95
disk 13	3.5	0.44	0.28
disk 14	224.9	0.0	0.0
disk 15	112.7	10.7	5.0
disk 16	7.8	3.4	0.0
disk 17	2.1	0.27	0.17
disk 21	11.3	1.4	0.93
disk 22	3.7	0.47	0.30
disk 23	1.8	0.34	0.06
disk 24	66.1	4.9	1.3
disk 25	172.9	0.0	0.0
disk 26	17.0	2.2	1.4
disk 27	28.8	3.7	2.4
disk 28	0.0	31.7	2.9
disk 29	281.1	3.4	3.7

* in millions of instructions

Resource Attributes

The hardware resources represented in the system model are the CPU, the channels 1 and 2, and the disks connected to these channels. Only disks that are used in the benchmark are included in the model. The paging process, the swapping process, and the VIO facility are represented explicitly as software resources in the system model, but not in the queueing network model. Thus their resource attributes need not be determined.

The service discipline for the CPU is complicated [IBM76a] and will not be described here. It will be represented as a processor shared discipline in the computational model. The service disciplines for the channels and the I/O devices are first-come-first-served.

For each hardware component of the model, the service rate must be derived from the measurement data. This can be done according the following formula:

$$C(i) = \frac{Op(i)}{T(i)}$$

C(i): service rate of device i (operations per unit time)
Op(i): number of operations device i performed in the measurement interval
T(i): busy time of device i during the measurement interval

Since the overlap of disk and channel activity cannot be modeled directly in the computational model, the service rates of the disks are adjusted to represent only the non-overlapped part of the disk operations. To do this, the mean times of disk and channel operations are calculated individually. Then, the mean channel operation times are subtracted from the mean disk operation times. The inverse of these shortened disk operation times are used as service rates for the disks in the model. The inverse of the channel service times are used as the service rates of the channels.

To determine the CPU service rate, we also consider the high speed cache hit ratio, i.e., the proportion of the memory access requests that can be satisfied out of the high speed cache. This ratio can influence the CPU service rate considerably. The formula for determining the mean memory access time is:

$$Ia = Na * (Ic + Im * (1 - P))$$

Ia: mean memory access time per instruction
Na: mean number of memory accesses per instruction
Ic: access time to the high speed cache
Im: access time to main memory
P: high speed cache hit ratio

The mean execution time of an entire instruction can be calculated from the number of instructions executed and the CPU busy time. Then this time can be broken down into memory access time and actual execution time. The service rates are assumed to be class independent. This assumption may not hold in reality, as, for example, there is evidence that the high speed cache hit ratio is class dependent. The service rates used in the queueing network model are shown in table 4.

Table 4. Service rates of the hardware resources in operations per second.

CPU	2.07*10E6	disk 21	32.20
channel 1	84.65	disk 22	51.56
channel 2	99.22	disk 23	27.80
disk 11	47.58	disk 24	21.78
disk 12	31.74	disk 25	36.60
disk 13	18.80	disk 26	54.45
disk 14	37.56	disk 27	39.76
disk 15	33.96	disk 28	55.53
disk 16	23.43	disk 29	70.11
disk 17	16.07		

5.2.2 System Model of the Birth-Death Model

Program Behaviour Model

The program behaviour model for the birth-death model is very simple. The tasks circulate between the terminals and the computer system. The mean think time is the only parameter that must be specified. The terminal simulator of the benchmark has a mean think time of 12 seconds. The service demand is one interaction per cycle.

Interference Pattern

The swapping of temporarily inactive tasks and the load balancing could be modeled as dynamic interference. However, to keep the model simple enough to be solved analytically, this will not be done. All overhead is covered by the network model on the dispatcher level, so no interference pattern has to be specified for the birth-death model.

Workload Model

Since no interference is considered, the workload model is identical with the service requests by the users described in the program behaviour model.

Multiprogramming Mix

The parameter that corresponds to the multiprogramming mix in the birth-death model is the mean number of active terminals. Another related parameter is the maximum multiprogramming level, which restricts the number of TSO interactions resident at one time. The maximum number of active terminals in the benchmark is 25. Due to end effects within the TSO stream of the benchmark, the mean number of active terminals is 23. The maximum multiprogramming level for TSO interactions cannot be determined since it is dependent on the activity of the other job classes. Considering the large backlog of TSO interactions that is indicated by the large average number of ready TSO transactions in the measurement data, the average multiprogramming level must be close to the maximum multiprogramming level. The average multiprogramming level during the TSO class elapsed time is calculated from the resident times to be 2.45. The assumption of constant average multiprogramming levels for the other classes does not hold in practice because the SRM balances the overall load in the system, so that the multiprogramming levels of all classes are highly interdependent. However, we do not have any information about this relationship, so we assume constant multiprogramming levels. If a joint probability distribution of the multiprogramming mix could be sampled, we could circumvent this approximation.

Resource Attributes

The server of the birth-death model represents the entire computer system. The SRM admits jobs into the multiprogramming mix according to an algorithm that is designed to balance the overall load of the system. However, in the benchmark all TSO tasks have

the same characteristics, so the FCFS discipline is a good approximation. The service rates of the birth-death model are the load dependent throughput rates that are determined by the queueing network model. The TSO throughput rates of the queueing network model are weighted to obtain the service rates for TSO interactions, the birth-death model assuming multiprogramming levels of 2.1 for GPJS and 0.73 for HSJS.

6. *COMPUTATIONAL MODELS*

6.1 Concepts

Analytic queueing network models are too complex a subject to be discussed in detail here. For an overview see, for example, the September 1978 issue of *ACM Computing Surveys* or [KIEN77]. In the following only a very brief overview is given.

An important distinction among queueing network models is whether or not the so-called "local balance" assumptions are satisfied [BCMP75]. Models in local balance can be solved by relatively inexpensive convolution algorithms [RK76], and, as mentioned earlier, they require only summary statistics as input. However, they impose some severe restrictions on the models: at servers with FCFS service discipline, all service time distributions must be assumed to be exponential with the same mean for all classes, and priority service disciplines cannot be represented.

Models not assuming local balance avoid these restrictions but they require much more detailed measurement data (e.g., interevent time distributions), and the systems of linear equations that must be inverted for their solution reach easily 100000 equations with 100000 unknowns. Thus, this method is too expensive in most realistic cases.

A number of approximation methods have been developed that, at least partially, overcome the problems of local balance models and whose cost is in the range of the cost for local balance methods. There is an iterative approach that facilitates the modeling of preemptive priority disciplines [SEVC77]. For the modeling of non-exponential service time distributions with FCFS service disciplines several approximation methods are available [CHW75, SLTZ77]. Finally, the diffusion approximation method [KOBA74a, KOBA74b, GP77] offers an approach using a set of differential equations similar to the diffusion equations in physics. This approach can handle FCFS service disciplines, arbitrary service time distributions, and several classes of customers.

6.2 Computational Multiple Class Queueing Network Model

For the queueing network model, a multiple class model in local balance is chosen, so the convolution algorithm can be applied. Only hardware resources will be represented explicitly in the queueing network model. The software service facilities like paging and swapping will be represented only implicitly as service demands on the hardware resources. Since the processor-shared discipline is used for the CPU, class dependent CPU

service time distributions can be assumed (although, they need not be specified). For the I/O subsystem, servers for the channels 1 and 2 and the disks are modeled, assuming FCFS disciplines and class independent exponential service time distributions. The spooling devices and channel 0 are not represented in the model since the spooling activity is asynchronous to the rest of the system's operation. The spooling overhead, however, is included in the workload matrix. The usage of the CPU, the disk channels, and the disks for the spooling system, however, are captured in the model. The structure of the model is shown in figure 4.

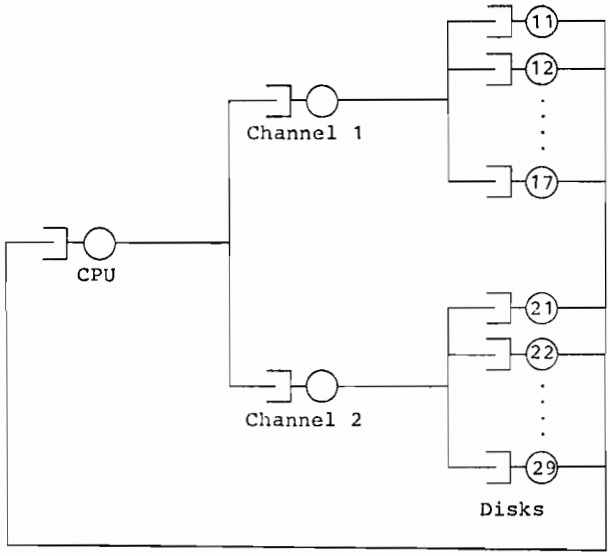


Figure 4. Queueing network model.

Table 5. Utilizations and throughputs of the multiple class queueing model.

	utilizations			throughput (jobs per second)		
	CPU	CH 2	CH 3	GPJS	TSO	HSJS
model results	.849	.281	.369	57.3	1042	303
measurement data	.820	.269	.346	56	950	308
relative error (%)	+3.6	+4.4	+6.7	+2.4	+9.7	-1.6

The input parameters required are the multiprogramming mix and the workload matrix with each entry divided by the service rate specified the corresponding device.

In order to obtain the performance measures for the average multiprogramming mix, the model is evaluated under the neighbouring integral multiprogramming mixes, and the results are linearly interpolated. The sequence in which the classes are chosen for interpolation does not affect the results.

The results of the model together with the corresponding measurement data and the relative errors where applicable are shown in tables 5 and 6. The disk performance measures are not listed. They are not very meaningful due to the special calculation of the disk service rates. Note that the mean queue lengths shown do not add up to the average multiprogramming levels of their respective classes because the mean queue length at the disks must be included in such a sum. The mean resident times can be calculated as the inverse of the throughput times the multiprogramming level of the respective class.

GPJS 81.5 secs
HSJS 3.4 secs

The response time of the TSO interactions will be determined by the birth-death model.

Table 6. Mean queue lengths of the multiple class queueing network model.

	mean queue lengths			
	GPJS	TSO	HSJS	total
CPU	1.055	0.653	0.417	2.125
Channel 2	0.202	0.126	0.038	0.366
Channel 3	0.163	0.322	0.029	0.514

These results seem to indicate that the overall resource usage is represented more or less accurately, but that too much of the overhead is attributed to HSJS and too little to TSO. The general breakdown ratio is a critical parameter for this model. A small experiment whose results are shown in table 7 can prove this. The first ratio is derived by distributing the unaccounted resource usage only according to the I/O related data. Apparently, too much overhead is attributed to GPJS, and not enough to TSO and HSJS. A shift of 10% of the unaccounted overhead in the ratio from GPJS to TSO improves the results for GPJS and TSO, the error for HSJS becoming even larger. The third ratio, derived by the rationale described in the section on the interference pattern, results mainly in a shift of overhead from GPJS to HSJS. The TSO error is only slightly changed. As a consequence of this experiment, we know that the distribution of unaccounted resource usage must be performed with great care. The ideal case would certainly be if all resource usage were accounted to user tasks by the monitors. Where this is not possible, the breakdown ratio can be adjusted until the the utilization and the throughput values of the model match the measured values as well as possible. This

calibration must then be validated by a model based on a new set of measurement data collected under a different workload.

Table 7. Error calculations for different breakdown ratios.

breakdown ratio			relative throughput error (%)		
GPJS	TSO	HSJS	GPJS	TSO	HSJS
.465	.508	.027	-12.7	+23.5	+21.9
.365	.608	.027	-6.9	+9.5	+23.0
.276	.600	.124	+2.4	+9.7	-1.6

6.3 Computational Birth-Death Model

The parameters of the system model can be used directly as input to the solution algorithm, so that no transformation need be performed for the parameter representation. For the structure of the model see figure 5.

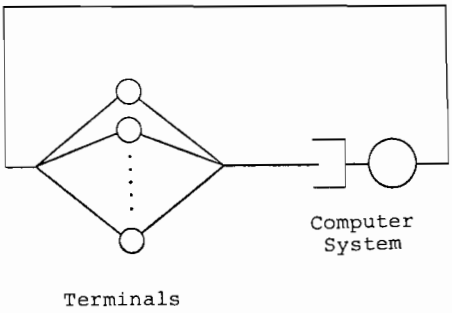


Figure 5. Birth-death model.

To obtain the service rates for the birth-death model, the queueing network model is evaluated for TSO multiprogramming levels of 0 to 3 and all the multiprogramming levels of the other classes required for the interpolations to obtain the TSO throughput values with average mutliprogramming levels of 2.1 for GPJS and 0.73 for HSJS. The birth-death model then is evaluated for the maximum multiprogramming levels of 2 and 3. The results, the average number of transactions in the system and the throughput of transactions, are again linearly interpolated to obtain the values for the fractional average multiprogramming level of TSO, 2.45. The results of the birth-death model are shown in table 8.

Table 8. Results of the birth-death model.

MPL	E[n]	throughput		response time	
		model (per sec)	rel.error (%)	model (sec)	rel.error (%)
2.4	15.92	.556	+1.4	28.64	-5.9
2.5	16.12	.573	+4.6	28.14	-7.6
2.6	16.33	.590	+7.7	27.67	-9.1

In order to compare the model results with the measurement data, the mean response time is calculated from the measurement data (following some operational considerations [BUZE76]) to be 30.45 seconds. Considering this response time and a mean think time of 12 seconds, the benchmark certainly does not represent a very realistic TSO load. This was discovered only during the modeling study. Dividing the number of interactions by the TSO class elapsed time, we obtain a mean throughput value of 0.548 interactions per second.

There are several reasons for the errors in the model results. Because the TSO throughput in the queueing network model is too high, the throughput of the birth-death model must also be too high. The maximum multiprogramming level in the system is not fixed as it is in the model. The state probabilities of the birth-death model show that for all cases evaluated the probability that the multiprogramming level is at its maximum is greater than 0.99. This finding is consistent with the observation of the large backlog of TSO tasks in the system. So the multiprogramming level for TSO tasks in the system follows the multiprogramming level as specified by the SRM, and is not determined by the arrival stream and the service rate of the server as is asssumed by the birth-death model. Considering this observation, it is questionable whether the birth death model is needed at all.

7. *CONCLUSIONS*

The proposed modeling procedure is intended to provide a better understanding of the system and the modeling process. By stating the intermediate models explicitly, the procedure allows a more formal parameter transformation from the measurement data to the input parameters of the queueing network models. The isolation of the workload and the system's components in the system model makes the modeling decisions more explicit and exposes the influence of the different parts of the system on its performance.

To benefit fully from this procedure, a modeling study should be based on a large set of experiments under a wide range of workload conditions. Using detailed measurement data, sophisticated system models can be developed to serve as a basis for detailed modeling studies.

Four points that deserve more attention are: 1) Based on measurement experiments varying the load, interference functions could be established that implicitly reflect cause

and effect relationships of the operating system's overhead and thus put the interference pattern on a sound basis. 2) The load and class dependency of the CPU and I/O service rates could be determined. 3) With several sets of measurement data from actual operation of the system, a broadly based workload model could be developed that would reflect an average load situation of the system much better than the benchmark. 4) An independent set of measurement data would allow the validation of the modeling decisions.

ACKNOWLEGDEMENTS

The research for this paper greatly benefitted from discussions with Scott Graham, Ed Lazowska, Allan Levy, Satish Tripathi, and John Zahorjan, all of Project SAM 1976 - 77. John Sutherland of the UTCC was helpful in providing the measurement data. Good comments on the presentation were provided by Vicente Aragon, Tony Chu, Lenny Freilich, and Ben Welleschuk, members of Project SAM 1977 - 78. This study was financially supported by the Deutscher Akademischer Austauschdienst and the Department of Computer Science of the University of Toronto.

References

[BARD77]
Y. Bard. The Modeling of Some Scheduling Strategies for an Interactive Computer System. International Symposium on Compter Performance Modeling, Measurement, and Evaluation, Yorktown Heights, 1977.

[BCMP75]
F. Baskett, K. M. Chandy, R. Muntz, F. Palacios. Open, Closed, and Mixed Networks of Queues with Different Classes of Customers. Journal of the ACM, 22, April 1975.

[BUZE76]
J. P. Buzen. Fundamenal Laws of Computer Performance. International Symposium on Computer Performance Modelling, Measurement, and Evaluation, Cambridge, March 1976.

[BUZE78]
J. P. Buzen. A Queueing Network Model of MVS. Computing Surveys, September 1978.

[CHW75]
K. M. Chandy, U. Herzog, L. Woo. Approximate Analysis of Queueing Networks. IBM Journal of Research and Development, January 1975.

[CM73]
R. Cavanagh, G. Milandre. Hardware Monitoring at the University of Toronto Computing Centre. Internal UTCC document, May 1973.

[DB78]
P. Denning, J. P. Buzen. The Operational Analysis of Queueing Network Models. Computing Surveys, September1978.

[GIAM76a]
T. Giammo. Validation of a Computer Performance Model of the Exponential Queueing Network Family. International Symposium on Computer Performance Modeling, Measurement, and Evaluation, Cambridge, 1976.

[GIAM76b]
T. Giammo. Extensions to Exponential Queueing Network Theory For Use in a Planning Environment. Proceedings of Compcon '76 Conference, IEEE, September 1976.

[GP77]
E. Gelenbe, G. Pujolle. A Diffusion Model for Multiple Class Queueing Networks. Rapport de Recherche No 242, Laboratoire de Recherche en Informatique et Automatique, 1977.

[IBM76a]
OS/VS2 System Programming Library, Initialization and Tuning Guide, IBM Form No. GC28-0755

[IBM76b]
OS/VS2 MVS Resource Measurement Facility (RMF), Reference and User's Guide, IBM Form No. SC28-0740

[IBM77]
OS/VS2 MVS System Programming Library, System Management Facilities (SMF), IBM Form No. GC28-0706

[KGT77]
A. Krzesinski, S. Gerber, P. Teunissen. A Multiclass Network Model of a Multiprogramming Timesharing Computer System. Proceedings IFIP congress '77, Toronto, 1977.

[KIEN77]
M. Kienzle. Measurements of Computer Systems for Queueing Network Models. M.Sc. Thesis, University of Toronto, Technical Report CSRG-86, Computer Systems Research Group, University of Toronto, 1977.

[KOBA74a]
H. Kobayashi. Application of the Diffusion Approximation to Queueing Networks, I: Equilibrium Queue Distributions. Journal of the ACM, 21, 1974.

[KOBA74b]
H. Kobayashi. Application of the Diffusion Approximation to Queueing Networks, II: Non-Equilibrium Distributions and Applications to Computer Modelling. Journal of the ACM, 21, 1974.

[KR75]
H. Kobayashi, M. Reiser. On Generalization of Job Routing Behaviour in a Queueing Network Model. Research Report RC 5252, Yorktown Heights, 1975.

[RK76]
M. Reiser, H. Kobayashi. On the Convolution Algorithm for Separable Queueing Networks. International Symposium on Computer Performance Modelling, Measurement, and Evaluation, Cambridge, March 1976.

[ROSE76]
C. Rose. Validation of a Queueing Network Model with Classes of Customers. International Symposium on Computer Performance Modelling, Measurement, and Evaluation, Cambridge, 1976.

[SEVC77]
K. C. Sevcik. Priority Scheduling Disciplines in Queueing Network Models of Computer Systems. Proceedings IFIP Congress '77, Toronto, 1977.

[SLTZ77]
K. C. Sevcik, A. Levy, S. Tripathi, J. Zahorjan. Improving Approximations of Aggregated Queueing Network Subsystems. International Symposium on Computer Performance Modelling, Measurement, and Evaluation, Yorktown Heights, 1977.

[SU78]
K. C. Sevcik, D. Unruh. A Case Study in Predicting TSO Response Times. Submitted for Publication.

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

Synchronization Problems in Hierarchically Organized Multiprocessor Computer Systems

U. Herzog, W. Hoffmann

Institute of Mathematical Machines and
Data Processing (III)

University of Erlangen-NÜrnberg, Germany

The evolution of technology provides the opportunity of connecting inexpensive processors to build medium or large scale computer systems. However, operating such systems, serious coordination problems (synchronization, data and code sharing, etc.) may occur.

Characteristic performance values, such as throughput, mean response time, and distribution functions are derived directly for fundamental computer structures and operating modes. More complex systems are investigated by a hierarchical modeling technique.

INTRODUCTION

Multiprocessor computer systems with two or three processing units have been built since many years. Due to inexpensive hardware-components and minicomputers there is an increasing interest in building systems with some ten or even hundreds of processors [5-7].

Rather than running independent tasks on different processors one also tries to take advantage of the parallelism inherent in many problems, i. e. application programs are decomposed into sets of parallel cooperating subtasks and processed in parallel, when possible [11,15]. So we may increase not only the throughput of a system: run-times (and therefore response-times) for individual application programs may be reduced significantly, too. Then, however, difficult coordination problems (synchronization between tasks, data and code sharing, etc.) may occur [2,5-7,11,15,21].

In this paper we investigate synchronization problems and their influence on performance by a new class of queuing models.

We first describe in outline the architecture of hierarchically organized multiprocessor computer systems with centralized control (our modeling technique may be applied, at least partially, also for systems without centralized control since we may find there master-slave relations, too).

The timely sequence of events is determined not only by the structure and operating mode of a multiprocessor system. It is heavily influenced also by the internal structure of the application programs to be run on the system.

We therefore classify application programs in section 3, think about their implementation on hierarchical systems, and develop the corresponding queuing models.

Section 4 deals with the analysis of two level hierarchies under Markovian and Non-Markovian assumptions.

Section 5 summarizes the results for a three level multiprocessor system on which a traveling salesman algorithm runs as application program.

We then conclude, section 6, with some remarks on ongoing and future research for multiprocessor computer systems.

ARCHITECTURE OF HIERARCHICALLY ORGANIZED COMPUTER SYSTEMS

1. General remarks

Hierarchical structures have been applied successfully in many organizations, industries and technical systems. It therefore seems to be reasonable to use this type of organization for multiprocessor systems, too. Hierarchical structures are transparent since we may distinguish clearly between organizational and application work and it is possible to concentrate coordination problems while distributing independent user tasks.

Typical examples are the EGPA-project [6], the multiprocessor system at the SUNY [7], the Siemens-system SMS [17], MOPPS [20], X-TREE [4] and others.

Because our investigations were initiated and influenced by the EGPA-project, we outline next the EGPA-architecture.

2. The EGPA-pyramid [6]

The EGPA (Erlangen General Purpose Array) consists of a rectangular array of processors (A-processors) connected via multiport memories. Each processor may access its

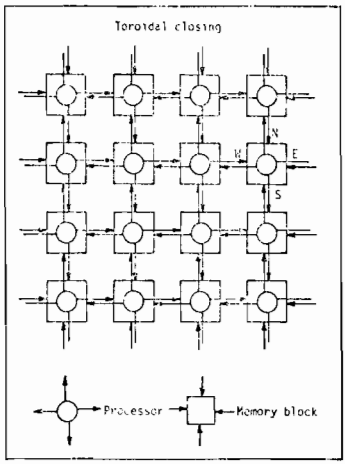


Fig. 1: Erlangen General Purpose Array (EGPA)

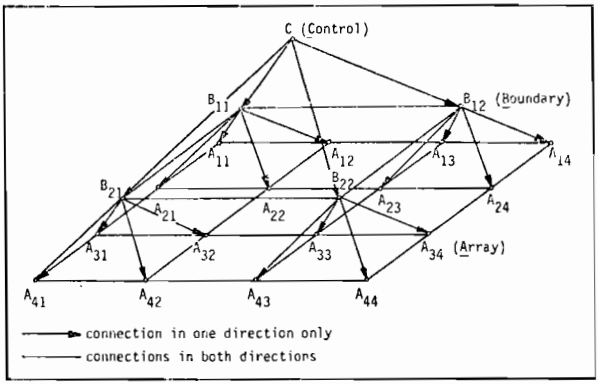


Fig. 2: Cellular structure of EGPA

"own" memory, and the memories of its four neighbours in the north, west, south and east direction (figure 1). The edges of the rectangle are connected to form a toroid. In addition to the array-processors, there are processors dedicated to the transfer of data between the array-processors and the peripherals.

At present, each of these processors, called boundary (B)-processors, is allocated to a 2 x 2 square of the array. In addition there is a single processor, called control (C)-processor, which has a supervisory function. The pyramid structure is shown in figure 2. A pilot-pyramid is being implemented now.*

PARALLELISM, SYNCHRONIZATION PROBLEMS AND MODELING

1. General remarks

The flow of information in a multiprocessor computer system depends on the hardware-components, the interconnection scheme and the operating system. It is, however, heavily influenced also by the internal structure of the algorithms implemented in the application programs.

Following the work of Adams [1] and others, we describe a program by means of a directed graph, the nodes representing subtasks (well defined functions or sets of functions), the edges showing interdependencies and representing data buffers (unlimited FIFO-queues). Nodes (subtasks) are performed if and only if each input edge to this node contains at least one data. We classify several types of parallel algorithms and develop equivalent queuing models for hierarchical multiprocessor computer systems.

2. Type-1-program structure

The program consists of a loop which may be passed several times, cf. figure 3. Within that loop n independent subtasks can be distinguished (there may exist some pre- and postprocessing). A new loop-cycle may be started iff** all n independent subtasks are completed.

---

* The EGPA-Project is supported by the BMFT, the German Ministry of Research and Technology.

** iff = if and only if

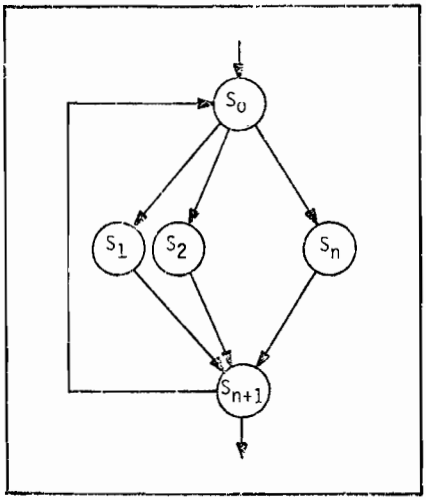


Fig. 3: Type-1-program structure (s_i-subtasks, cf. 3.2)

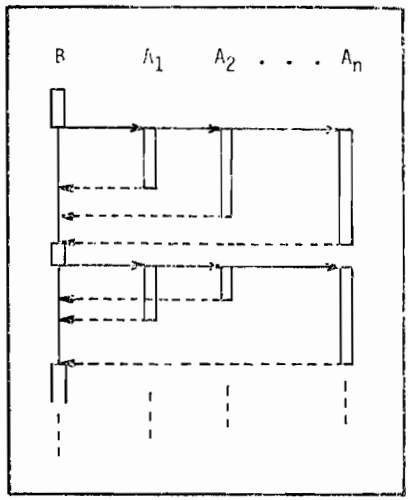


Fig. 4: Execution of a type-1-program on a two-level hierarchy.

Problems are often of this type: algorithms for the solution of linear-algebraic or partial differential equation systems, optimization procedures, simulations including subruns for the purpose of estimating confidence intervals, problems of picture processing, etc. etc.

Such algorithms may be implemented very efficiently on a hierarchically organized multiprocessor system with two levels (cf. figure 4):

- At first the source program is translated, loaded and then started by the B-processor (abbreviations, see section 2).
- The B-processor then initiates the execution of n independent subtasks by the A-processors.
- Having completed its subtask, each A-processor has to inform the B-processor.
- Postprocessing and preparation of a new loop-cycle by the B-processor is only possible when all subtasks are completed.

Queuing models which allow to describe and analyze the traffic flow including the above synchronization problem are shown in figures 5 to 9 (synchronization is shown symbolically by lying brackets).

- Multiprogramming

We first assume multiprogramming for both B- and A-processors (cf. figure 5):

Newly arriving demands (source programs) are processed by the B-server (B-processor), then it generates n independent sub-demands and distributes them simultaneously among all n A-servers (more sophisticated transfers, cf. section 6).

Sub-demands may have to wait since A-servers may be busy at that time. After completion each sub-demand is buffered in the corresponding input queue of the B-server.

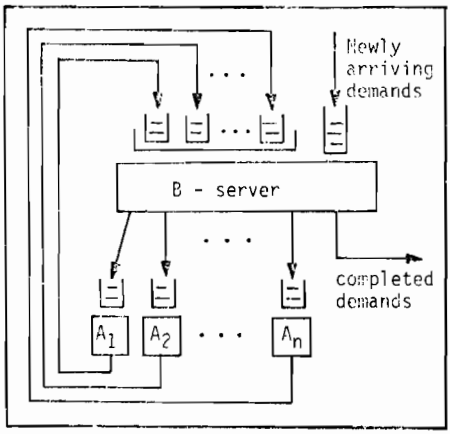


Fig. 5: Open model

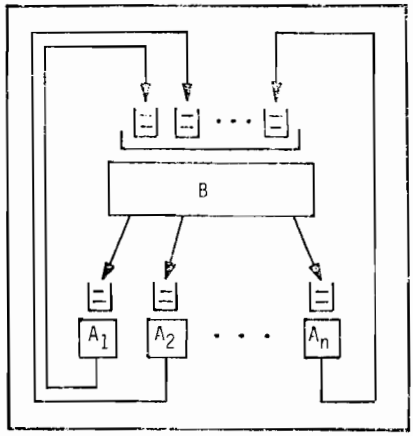


Fig. 6: Closed model

Iff all n sub-demands, belonging to a specific demand, are buffered, and the B-server is empty, they are removed simultaneously (symbolized by ⌊⌋) from the n parallel input queues and processed in one step. After completion there are two possibilities

- the (complete) demand leaves the system, or
- n new sub-demands are generated simultaneously and a new cycle is started.

Figure 6 shows the corresponding closed queuing model. It allows to model systems with a constant degree of multiprogramming. As we shall see in section 4, closed queuing models are the basic models because open models may be analyzed via these models.

- Monoprogramming

Multiprogramming allows to increase system throughput. However, for reason of simplicity and transparency of the operating system there is a trend to introduce monoprogramming, again [14,21,22,24].

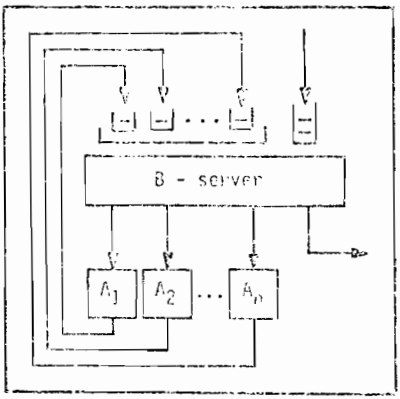


Fig. 7: Open model for monoprogramming

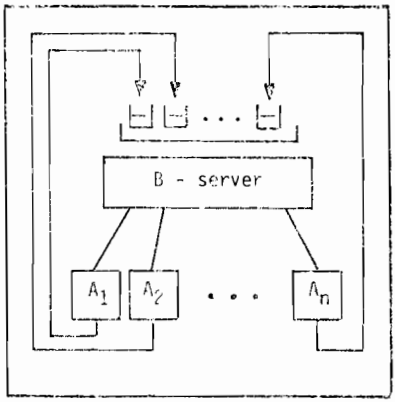


Fig. 8: Closed model for monoprogramming

The corresponding open and closed queuing models may be derived readily according to the above section and are shown in figures 7 and 8, respectively: no queues build up in front of the A-servers!

Obviously the utilization of the B-server is rather poor. There are, however, two possibilities to avoid this lack:

1) The B-server services also subtasks,
2) A mixed multi- and monoprogramming mode is introduced.

Since the analysis of solution 1 is very similar to that of monoprogramming we now discuss the second possibility.

- Mixed multi- and monoprogramming

Multiprogramming for the B-processor and monoprogramming for the A-processors seem to be a reasonable solution for many applications. Figure 9 shows the corresponding (closed) queuing model and is rather self-explanatory:

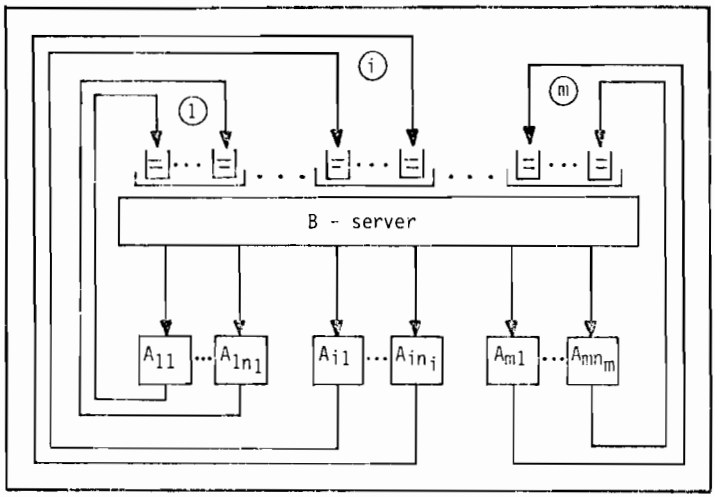


Fig. 9: Closed queuing model for mixed mode

Be given a number m of independent demands (t_1 ..., t_i ..., t_m) to be served sequentially (!) by the B-server. After completion each task t_i (i = 1, ..., m) generates n_i independent subtasks to be processed by the reserved A-processors A_{i1} to A_{in_i}. Task t_i may be started again if and only if all subtasks have been completed by the A-processors. If the B-server is busy, complete demands wait in front of the server and are served in the order of arrival (FIFO).

Note, synchronization is only necessary between sub-demands belonging together, an important fact for analysis.

3. Type-2-program structure

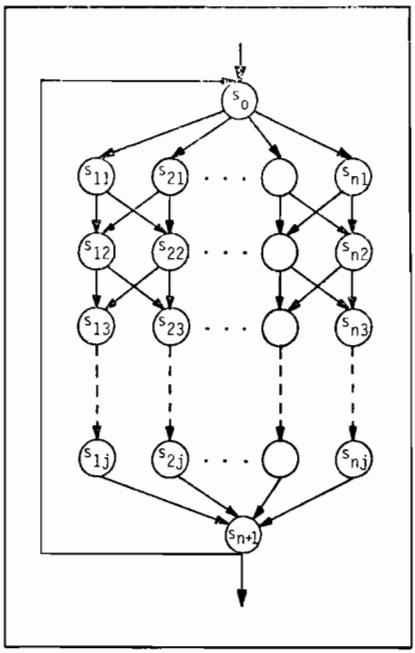


Fig. 10: Type-2-program structure

Here, the program also consists of a loop. However, the n subtasks influence each other in some way (figure 10 shows one possibility) rather than being completely independent.

So, in addition to the overall synchronization, a "local" synchronization of subtasks is also necessary.

The execution of a type-2-program structure may be performed on a hierarchical multiprocessor system as follows:

As before, the source program is translated, loaded and started by the B-processor. Again, the B-processor distributes n subtasks to the A-processors. However, after some processing low level (local) synchronization between several A-processors is necessary.

Processing continues and more low level synchronizations may occur until an overall synchronization by the B-processor is necessary.

Closed queuing models for monoprogramming are shown in figure 12, models for multiprogramming and mixed mode may be obtained accordingly (for reasons of simplicity we propose to use the simplified model rather than the detailed model which also shows the low level synchronization buffers).

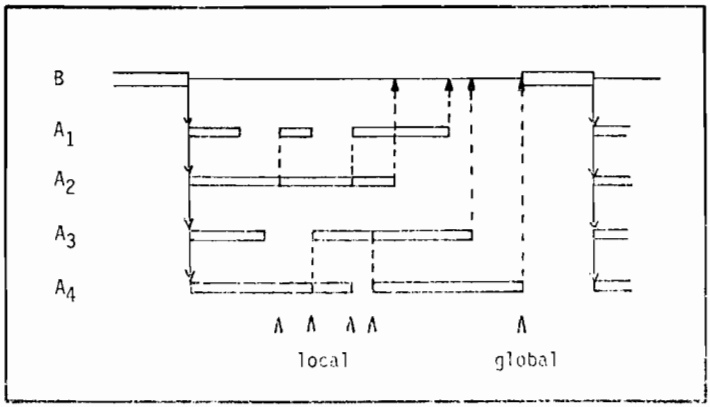


Fig. 11: Timing diagram with local and global synchronization

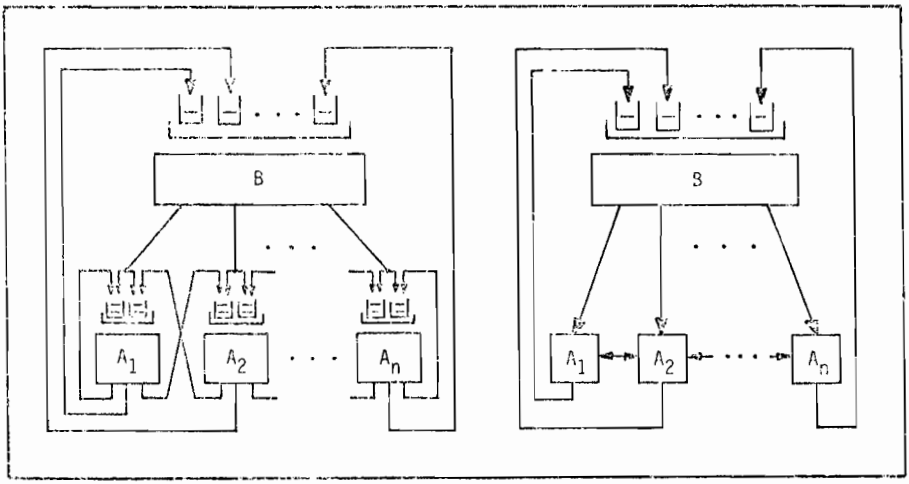


Fig. 12: Detailed (left) and simplified queuing models for systems with both global and local synchronization

4. Type-3-program structure

There are two reasons why the degree of parallelism (and therefore the number of necessary A-processors) may vary:

1) The changing number of independent subtasks is characteristic for many applications (inherent in the algorithm, cf. fig. 13)
2) The degree of parallelism of the algorithm is larger than - and not an integral multiple of - the number of A-processors available at the moment.

Figure 14 demonstrates the basic idea of modeling that type of problems: the B-processor always generates (!) n independent sub-demands. The A-servers are by-passed, however, with (different) probabilities $(1 - q_i)$, $i = 1, \ldots, n$. In other words: we always generate a constant number of sub-demands. For some of them (dependent on the task to be processed) there is zero processing time necessary at the A-processor.

It is possible to investigate such models under various assumptions [13]. And it is not too difficult to realize that models of this type allow to "simulate" exactly models where the B-processor generates i sub-demands according to an arbitrarily chosen probability distribution p_i $(i = 1, \ldots n)$.

(Figures 13 and 14 see next page)

5. Type-4-program structure

Completely independent tasks (fig. 15) may be run, of course, on a hierarchical system, too: each source program is translated and loaded individually by the B-processor and then processed by an A-processor.

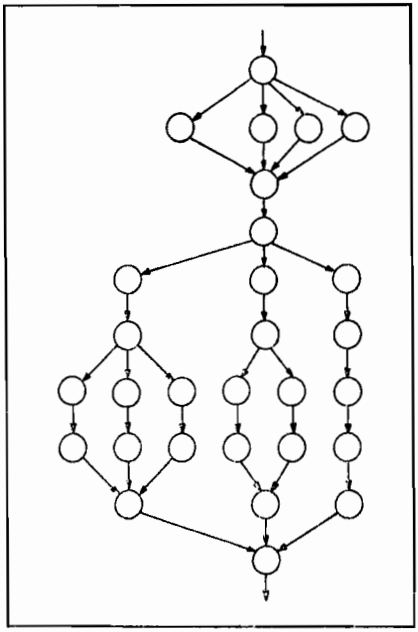


Fig. 13: Structure of an ALGOL program which generates a symmetric matrix [19].

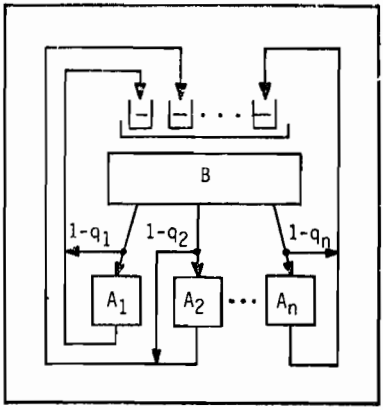


Fig. 14: Closed model for type-3-programs

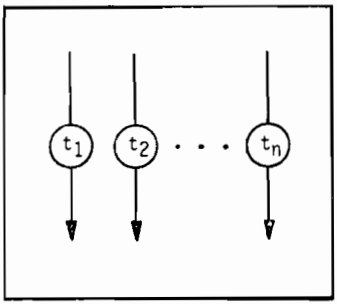


Fig. 15: Type-4-program structure (independent tasks).

Figures 16 and 17 show the corresponding open and closed queuing models (queuing may be or may not be allowed in front of the A-processors, dependent on the operating mode.)

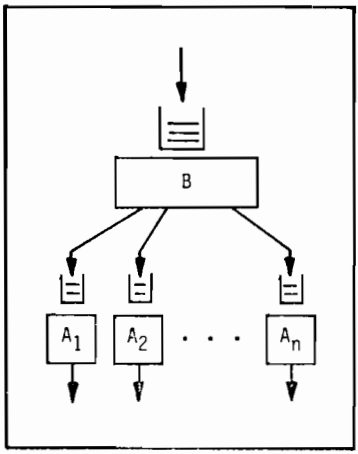


Fig. 16: Open model for type-4-programs

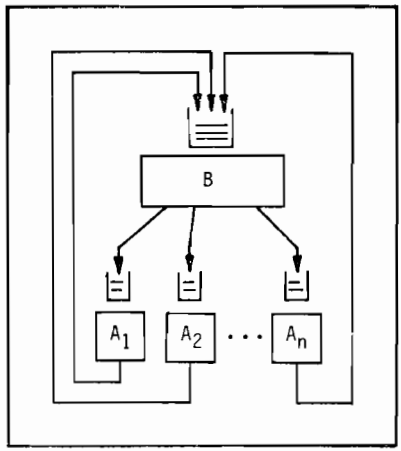


Fig. 17: Closed model for type-4-programs

It is interesting to note that the closed model presented in fig. 17 plays an important role when analyzing mixed models according to figure 9 (cf. section 4.2).

6. Mixture of program structures

Program structures presented above may occur in "pure form" only in some special purpose multicomputer systems. In the general case, however, programs of different structures have to be processed, and within one program the structure may also vary.

Then, the corresponding queuing model is a mixture of several pure models, in case of closed models mainly a combination of figures 9, 12 and 14.

Although we have not investigated yet such a model, its analysis seems to be straight forward. From our point of view it seems to be more difficult to find a characteristic workload and to choose the model parameters accordingly (cf. section 6).

PERFORMANCE ANALYSIS FOR TWO LEVEL HIERARCHIES

1. General remarks

All closed models presented above (and some open, too) have been investigated under various assumptions [8,9,12,13]:

- Exponentially distributed service times for both B- and A-servers, the mean value may vary for different A-servers,
- General service time distributions, mostly of Erlangian and hyperexponential type, respectively.

In analyzing the performance we usually assumed stationarity, determined then state probabilities and characteristic performance values such as

- system throughput
- server utilization individual for each server
- mean numbers of A-servers working simultaneously
- mean numbers of demands and/or sub-demands waiting in front of servers (queue length)
- mean synchronization time, i.e. mean time between the moment when all related sub-demands start and the last one is completed
- mean cycle time for complete demands, the sum of synchronization time and processing time at the B-server
- distribution functions for both the synchronization and cycle time.

Some results for the most important mean cycle time have been summarized in table 1, more detailed information on three closed models may be found in section 4.2.

Open models can be analyzed efficiently by a hierarchical modeling technique [10, 16], the results of which are often accurate approximations, sometimes exact [9].

Finally, we would like to point out that our investigations are strongly related to the problem of CPU-I/O-overlap [3,18,23]. So the multiple CPU-I/O-problem may be solved now, too.

2. The closed model for type-1-program structure and mixed multi- and monoprogramming.

Be given a model structure according to figure 9, the service discipline of which being described in section 3.2. Service times for both B- and A-servers are assumed to be exponentially distributed with

μ : service rate for the B-server

λ_{ij} : service rate for the A_{ij}-server, $i \in \{1,2,\ldots,m\}$ $j \in \{1,2,\ldots, n_i\}$

m : number of complete demands D_i (competing for the B-server), i.e. degree of multiprogramming.

n_i : number of parallel sub-demands belonging to demand D_i.

Analysis is performed under equilibrium conditions, i.e. stationarity is assumed.

- Decomposition: Recall, again, that synchronization is only necessary between sub-demands S_{ij}, $j \in \{1,2, \ldots n_i\}$ belonging together. So, if we are able to analyze all individual synchronization processes, the overall behaviour is described exactly by the following mode, known from figure 17:

B-server: service times exponentially with rates μ as in the complete model.
A-server: service times generally distributed with different service rates λ_i' according to the d.f. of the synchronization times $F_i(t)$ and its mean $E_i[T_S]$, $i \in \{1,2, \ldots m\}$. No queues in front of the A-servers.

We therefore attack at first the (individual) synchronization problem also being the solution for the monoprogramming model shown in figure 8. Then, in a second step, we briefly outline the solution for the multiprogramming model presented in figure 17, a generalization of the G/M/1/m-model.

Type-1-program models (monoprogramming) — $n \neq 2$

a) exponentially distributed service times in the A-processors

$$t_z = \frac{1}{\mu} + \frac{3}{2\lambda}$$

b) Erlang - k - distributed service times in the A-processors

$$t_z = \frac{1}{\mu} + \frac{1}{\lambda} + \frac{1}{2k\lambda} \sum_{i=0}^{k-1} \frac{(k-i)\cdot \prod_{\ell=1}^{i} (\ell + k - 1)}{i!\, 2^{k+i-1}}$$

c) H_2 - distributed service times in the A-processors (mean service time: $\frac{\alpha_1}{\lambda_1} + \frac{\alpha_2}{\lambda_2}$)

$$t_z = \frac{1}{\mu} + \frac{2\lambda_1\lambda_2 + 3(\alpha_1\lambda_2^2 + \alpha_2\lambda_1^2) + (\lambda_2\alpha_1 + \lambda_1\alpha_2)(\lambda_1\alpha_1 + \lambda_2\alpha_2)}{2\lambda_1\lambda_2(\lambda_1 + \lambda_2)}$$

n arbitrary

a) exponentially distributed service times in the A-processors

$$t_z = \frac{1}{\mu} + \frac{1}{\lambda} \sum_{i=1}^{n} \frac{1}{i}$$

Type-1-program models (mixed multi- and monoprogramming) — m arbitrary

(cf. section 4.2)

$$t_z^{(i)} = E_i[T_s] + \frac{\mu^{m-1} + \sum_{j=2}^{m} j \cdot \mu^{m-j} \cdot d_{m,i}(j)}{\mu\left(\mu^{m-1} + \sum_{j=2}^{m} \mu^{m-j} \cdot d_{m,i}(j)\right)}$$

$$d_{m,i}(j) = \sum_{\substack{i_1,\ldots,i_{j-1} \in \{1,2,\ldots,m\}\cdot\{i\} \\ \text{in twos different}}} \prod_{k=i_1,\ldots,i_{j-1}} \frac{1}{E_k[T_s]}\,; \qquad E_i[T_s] = \frac{1}{\lambda_i} \sum_{j=1}^{n_i} \frac{1}{j}$$

in the case of exponentially distributed service times ($\lambda_{ij} = \lambda_i$ for $1 \leq j \leq n_i$)

Type-2-program models — n arbitrary

a) exponentially distributed service times in the A-processors

$$t_z \leq \frac{1}{\mu} + \frac{s+1}{\lambda} \sum_{i=1}^{n} \frac{1}{i}$$

dependent on the specific local synchronization mode

(s: number of local synchronizations)

Type-3-program models — n arbitrary

a) exponentially distributed service times in the A-processors

$$t_z = \frac{1}{\mu} + \frac{1}{\lambda}\left[\prod_{j=1}^{n} q_j \left(\sum_{k=1}^{n} \frac{1}{k} + \sum_{k=1}^{n-1} \frac{1}{n-k} \sum_{i=1}^{k} \sum_{\substack{j_1,\ldots,j_i \in \{1,\ldots,n\} \\ j_1 < j_2 < \ldots < j_i}} \prod_{\ell=1}^{i} \frac{1-q_{j_\ell}}{q_{j_\ell}}\right)\right]$$

Type-4-program models — n arbitrary

(cf. section 4.2)

$$t_z^{(i)} = \frac{1}{\lambda_i} + \frac{\mu^{n-1} + \sum_{j=2}^{n} j \cdot \mu^{n-j} \cdot d_{n,i}(j)}{\mu\left(\mu^{n-1} + \sum_{j=2}^{n} \mu^{n-j} \cdot d_{n,i}(j)\right)}$$

with $d_{n,i}(j) = \sum_{\substack{i_1,\ldots,i_{j-1} \in \{1,2,\ldots,n\}\cdot\{i\} \\ \text{in twos different}}} \prod_{k=i_1,\ldots,i_{j-1}} \lambda_k$

Table 1: Mean cycle time t_z for different models (μ: rate for B-server, λ: rate for A-servers, q_i: branching probabilities (cf. fig. 14)).

- Synchronization: Let D_i be the complete demand the synchronization time of which has to be determined. Processing of all sub-demands S_{ij}, $j \in \{1,2,\ldots, n_i\}$, starts at the same instant. Suppose all processing times are exponentially distributed with uniform service rates $\lambda_{i1} = \lambda_{in_i} = \lambda_i$ (solution for different rates, cf. [8,13]).

Now, obviously, the interval T_{S_1} between the initiation of all sub-processes and the termination of the first sub-process is distributed exponentially:

$$(1) \quad P_i(T_{S_1} \leq t) = 1 - e^{-n_i \cdot \lambda_i t}, \quad t \geq 0$$

the interval T_{S_2} between the first and second termination according to

$$(2) \quad P_i(T_{S_2} \leq t) = 1 - e^{-(n_i-1) \cdot \lambda_i t}$$

etc.etc.

Therefore, the total synchronization time is (cf. fig. 18)

$$(3) \quad T_S = \sum_{j=1}^{n_i} T_{S_j},$$

and its distribution function is obtained by the n_i-fold convolution of exponential distributions with different mean values:

$$F_i(t) = P_i(T_S \leq t) = P_i(T_{S_1} \leq t) * P_i(T_{S_2} \leq t) * \ldots * P_i(T_{S_{n_i}} \leq t).$$

Applying Laplace transformation the following closed form solution may be obtained

$$(4) \quad F_i(t) = \sum_{j=1}^{n_i} (-1)^{j-1} \cdot \binom{n_i}{j} \cdot (1 - e^{-j \cdot \lambda_i t})$$

with mean and variance

$$(5) \quad E_i[T_S] = \frac{1}{\lambda_i} \cdot \sum_{j=1}^{n_i} \frac{1}{j}$$

$$(6) \quad VAR_i[T_S] = \frac{1}{\lambda_i^2} \sum_{j=1}^{n_i} \frac{1}{j^2}.$$

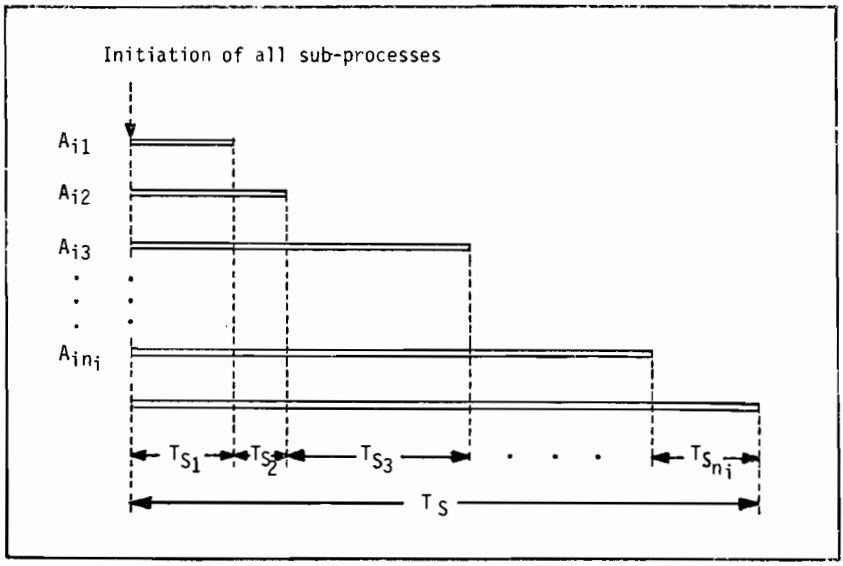


Fig. 18: Interpretation of the synchronization process and the way of analytic solution.

- Overall behaviour: Since the synchronization problem is now solved the overall behaviour is completely determined by the parallel-processing model presented in fig. 17. Recall, there is no queuing in front of the A-servers and the service rate of these m A-servers is

$$(7) \qquad \lambda_i' = \frac{1}{E_i[T_S]} \qquad i \in \{1,2, \ldots, m\}.$$

We first analyzed this model under Markovian assumptions: system states were described by a (m + 1)-dimensional vector, stationarity was assumed and the explicit solution derived for the state-probabilities, a generalization of the well known M/M/1/m-solution.

Secondly, we proofed the solution also being valid in case of general service time distributions for the A-servers. The detailed analysis may be found in [13], some characteristic performance values are:

Utilization of the B-processor Y_B

$$Y_B = \frac{1}{h} \left(\sum_{j=1}^{m} \sum_{i_1,\ldots,i_j \in \{1,\ldots,m\}} \mu^{m-j} \prod_{i = i_1,\ldots,i_j} \lambda_i' \right)$$

Utilization of each A-processor Y_A

$$Y_A = \frac{1}{h}\left(\mu^m + \sum_{j=1}^{m-1} \frac{m-j}{m}\,\mu^{m-j} \cdot \sum_{i_1,\ldots,i_j \in \{1,\ldots,m\}} \prod_{i=i_1,\ldots,i_j} \lambda_i'\right)$$

Mean cycle time t_z for a task with mean service time $\frac{1}{\lambda_i'}$ in the A-processor

$$t_z(i) = \frac{1}{\lambda_i'} + \frac{\mu^{m-1} + \sum_{j=2}^{m} j\,\mu^{m-j} \sum_{i_1,\ldots,i_{j-1} \in \{1,\ldots,m\}\setminus\{i\}} \prod_{k=i_1,\ldots,i_{j-1}} \lambda_k'}{\mu\left(\mu^{m-1} + \sum_{j=2}^{m} \mu^{m-j} \sum_{i_1,\ldots,i_{j-1} \in \{1,\ldots,m\}\setminus\{i\}} \prod_{k=i_1,\ldots,i_{j-1}} \lambda_k'\right)}$$

in twos different

with $$h = \mu^m + \sum_{j=1}^{m} \mu^{m-j} \sum_{i_1,\ldots,i_j \in \{1,\ldots,m\}} \prod_{i=i_1,\ldots,i_j} \lambda_i'$$

3. Numerical results

Figure 19 shows the mean cycle time for (complete demands) as a function of:

1) The program (and therefore model) structure: we may have one, two or four demands with four, two or one sub-demands respectively, and
2) The mean service rate μ of the B-server.

Furthermore, it is assumed that the service rate λ is uniform for all sub-demands. In order to compare the results for the same load per cycle of the B-server we introduced different scales. Note, however, that we compare various types of programs running on a given configuration!

PERFORMANCE ANALYSIS FOR A THREE LEVEL HIERARCHY

Figure 20 shows a three level multiprocessor computer system and the flow of information if we run an algorithm for the famous traveling salesman problem as application program on such a configuration (solution via 3 - optimal tours).

Rather than describing the details of the algorithm and its implementation [13] we focus our attention on the problem of modeling:

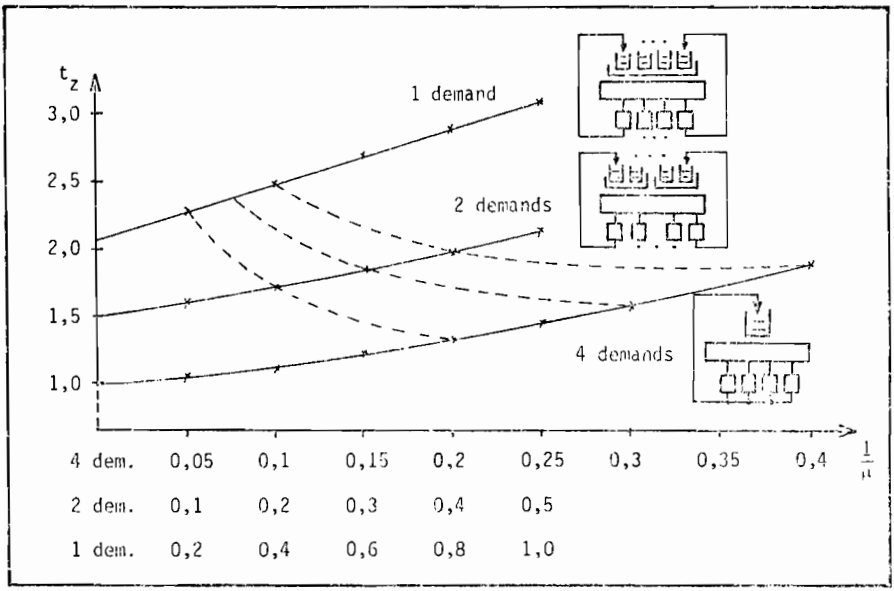


Fig. 19: Examples for the mean cycle time (cf. text).

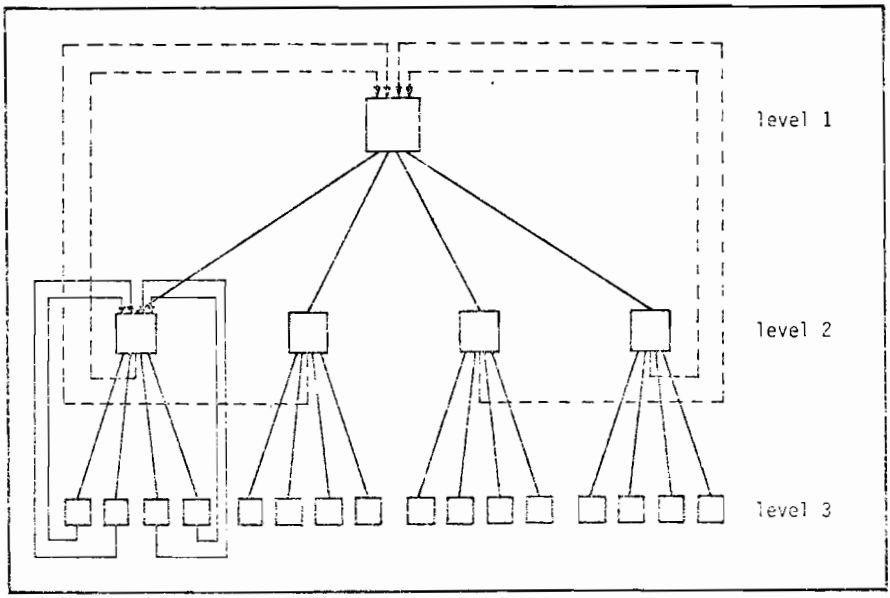


Fig. 20: Three level multiprocessor computer structure

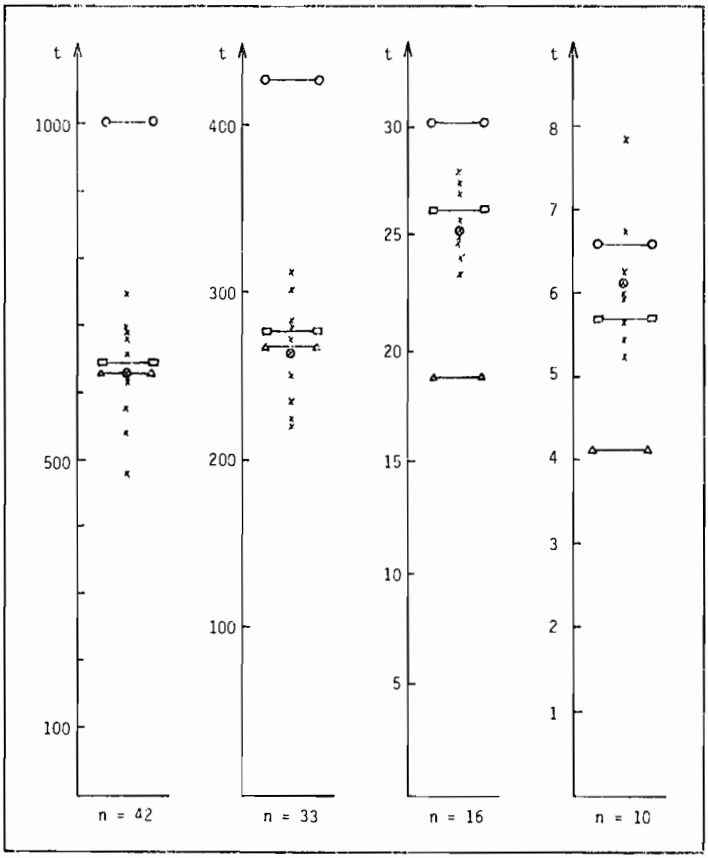


Fig. 21: Execution time t for the traveling salesman algorithm on a three level hierarchical multiprocessor computer system. Comparison between results from a step-by-step simulation of the algorithm (simulations x, mean value &) and three approximate solutions (proc. 1 o——o, proc. 2 △——△, proc. 3 □——□). Results are shown for n = 10, 16, 33 and 42 cities (cf. text).

It is rather unwise and probably unsuccessful to describe and analyze the complicated flow of information and all interdependencies by a single global queuing model. However, it was possible to describe the interactions between each level-2-processor and its level-3-processors by a parallel-processing model, shown in figure 17 and analyzed in section 4.2. On the other hand, interactions between the level-1-processor and all level-2-processors may be described accurately by the synchronization model, presented in figure 8.

So, applying the hierarchical modeling and analysis technique [10,16] we developed three approximate procedures. Some results are shown in figure 21 and compared with a step-by-step simulation of the complete 3-optimal-tour-algorithm (since the algorithm starts from a randomly chosen initial tour, the execution time varies even for the same number n of cities: we therefore performed the algorithm ten times for each problem).

Comparisons show that the (very simple) procedure 1 tends to overestimate the execution time. Procedure 2 yields, for a small number of cities, results which are somewhat too optimistic. Finally, the most sophisticated procedure 3 yields always accurate results.

CONCLUDING REMARKS

Introducing multiprocessor computer systems serious coordination problems (synchronization, data transfer, data- and load-sharing, etc. etc.) may occur.

In this paper we described and analyzed the effect of synchronization on system performance. In particular, taking into consideration the internal structure of programs, we developed specific models and analyzed them under Markovian and Non-Markovian assumptions as well.

Several directions for further work are apparent. Some of our models have only been investigated under Markovian assumptions, now we try to solve these problems more generally. More sophisticated models may be obtained if signalling overhead is introduced. Priorities are another interesting and important subject. It also seems to be possible to take into consideration the influence of data transfers on system performance. And, as mentioned above, the multiple CPU-I/O-overlap problem may be solved now, too.

Most significant is also the following task: we have to investigate the structure of various algorithms and programs in order to find a characteristic workload for parallel processing systems and their corresponding models.

In short, considering the evolution of technology and trends in hardware/software-development, there are still many important and challenging problems in the area of performance modeling and evaluation for multiprocessor computer systems.

REFERENCES

[1] ADAMS, D. A.:
A Model for parallel computations.
In: Parallel Processor Systems,Technologies, and Applications, Hobbs et al. (eds.), Spartan Books, New York, Washington (1970)

[2] BÄCHLE, A.:
Private communications on performance evaluation 1976/1977

[3] COTTON, L. W.; ABD - ALLA, A. M.:
Processing times for segmented jobs with I/O Computer overlap.
JACM 21, No. 1 (1974) pp. 18-30

[4] DESPAIN, A. M.; PATTERSON, D.:
X-Tree, A Tree Structured Multi-Processor Computer Architecture.
SIGARCH Newsletter, Vol. 6, No. 7 (1978) Computer Architecture News, pp. 144-151

[5] FULLER, S. et al.:
Multi-Microprocessors: An Overview and Working Example.
Proc. IEEE Vol. 66, No. 2 (1978) pp. 216-228

[6] HÄNDLER, W.; HOFMANN, F.; SCHNEIDER, H.J.:
A General Purpose Array with a Broad Spectrum of Applications.
In: Computer Architecture (Händler ed.), Informatik-Fachberichte 4, Springer-Verlag (1975) pp. 311-335

[7] HARRIS, J. A.; SMITH, D. R.:
Hierarchical Multiprocessor Organisations.
Computer Architecture News, Vol. 5, No. 7 (1977) pp. 41-48

[8] HERZOG, U.:
Verteilungsfunktion der Zykluszeit für das Synchronisationsmodell mit beliebiger Zahl von A-Prozessoren und einem Gesamtauftrag.
University of Erlangen-Nürnberg, unpublished memo, April 1978

[9] HERZOG, U.:
Synchronisations- und Zykluszeit für Typ-2-Programmstrukturen bei Monoprogrammierung, offene und geschlossene Warteschlangenmodelle.
University of Erlangen-Nürnberg, unpublished memo, July 1978

[10] HERZOG, U.:
Modeling of Data Networks.
Invited Paper at ITC (Intern. Teletraffic Congress) - Seminar on "Modeling of Stored Program Controlled Exchanges and Data Networks", Delft University of Technology, October 1977, to be published by North Holland

[11] HIBBARD, P.; HISGEN, A.; RODEHEFFER, T.:
A Language Implementation Design for a Multiprocessor Computer System.
SIGARCH Newsletter Vol. 6, No. 7 (1978) Computer Architecture News, pp. 66-72

[12] HOFFMANN, W.:
Queuing models for parallel processing and their application to a hierarchically organized multiprocessing system.
Paper accepted for First European Conference on Parallel and Distributed Processing, February 14-16, 1979, Toulouse, France

[13] HOFFMANN, W.:
Warteschlangenmodelle für die Parallelverarbeitung.
PH-D.-Thesis, University of Erlangen-Nürnberg 1978

[14] HOFMANN, F.; WENDLER, K.:
EGPA, Überlegungen zu Koordinierungsproblemen in Polyprozessoren mit begrenzten Nachbarschaften.
Presentation at the University of Erlangen-Nürnberg, July, 17, 1978

[15] JONES, A. K. et al.:
Programming Issues Raised by a Multiprocessor.
Proc. IEEE Vol. 66, No. 2 (1978) pp. 229-237

[16] KOBAYASHI, H.:
Modeling and analysis, an introduction to system performance evaluation methodology.
Addison-Wesley, 1978

[17] KOPP, H.:
Numerical Weather Forecast with the Multi-Microprocessor System SMS 201.
In: Parallel Computers - Parallel Mathematics, Feilmeier (ed.), IMACS, North Holland Publishing Company (1977) pp. 265-268

[18] MAEKAWA, M.; BOYD, D. L.:
Two models of task overlap within jobs of multiprocessing multiprogramming systems.
Proc. 1976 Int. Conf. on Parallel Processing, Wayne-State University and IEEE, pp. 83-91

[19] NETT, E.:
On further applications of the HU-algorithm to scheduling problems.
Proc. 1976 Int. Conf. on Parallel Processing, Enslow (ed.), Wayne-State-University, IEEE-ACM (1976) pp. 317-325

[20] SHIMOR, A.; WALLACH, Y.:
A Multibus-Oriented Parallel Processor System.
IEEE Trans., Vol. IECI - 25 (1978) pp. 137-141

[21] SIEWIOREK, D. P.:
Process Coordination in Multimicroprocessor Systems.
In: Microarchitecture of Computer Systems, Hartenstein and Zaks (eds.), Euromicro, North Holland Publishing Company (1975) pp. 1-8

[22] SIEGERT, H. J.:
Betriebsprogramme für dezentrale Rechensysteme.
NTG-Fachberichte, Bd. 62 (1978) pp. 207-213, VDE-Verlag, Berlin

[23] TOWSLEY, D. F.:
The effects of CPU: I/O overlap on computer system configurations.
SIGARCH Newsletter, Vol. 6, No. 7 (1978)

[24] WENDLER, K.:
Betriebssystemaspekte in hierarchisch modularen Polyprozessorsystemen - Modellierungsansätze und Koordinierungsmechanismen.
PH-D.-Thesis, University of Erlangen-Nürnberg (1978)

COMPUTATIONAL METHODS FOR QUEUEING NETWORKS

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

SOME EXTENSIONS TO MULTICLASS QUEUEING NETWORK ANALYSIS

Yonathan Bard
IBM Cambridge Scientific Center
Cambridge, Mass.

Mean value analysis of queueing networks is extended to obtain approximate formulas for several previously unsolved problems. Both finite population and large population asymptotic results are obtained. Among the cases treated are first-come-first-serve queues with different exponential service times for different user classes, dispatching priorities based on resource consumption, overlaps among different queues, blocking by constrained resources, and decomposable networks.

1. Introduction

Reiser and Lavenberg [1] have shown that queueing network problems can be solved very simply by means of relations involving only mean values of queue lengths, service times, and queueing times. The results are the same as those derived from product form solutions whenever the latter exist. Though the amount of computation required is the same, the mean value analysis is much simpler to derive and explain, and is numerically less troublesome since it does not require computation of very large normalizing constants. The new approach also leads to very simple derivations of previously known asymptotic formulas [2], and is capable of operational (in the sense of Buzen [3]) interpretation.

One of the main benefits of the mean value analysis is that it may be applied in a straightforward way to networks which do not have a product form solution. The results are generally correct only in the asymptotic case, but may provide good approximations in the finite case. Following a brief exposition of the method, we shall illustrate several such extensions.

2. Mean Value Analysis of Queueing Networks

We shall treat closed queueing networks with M user classes (chains) and K queues. We shall assume that the workload of each class i user is composed of a homogeneous set of work units, referred to as "transactions." Following Kobayashi and Reiser [4] we shall ignore (except where noted otherwise) the topology of the network, and assert that only the following quantities need be specified:

n_i = number of users in class i
t_{ij} = average total service time required by a class i transaction at queue j.

Let $N = \Sigma n_i$ be the total user population. We shall be interested in estimating the following quantities:

λ_i = class i transaction throughput.
N_{ij} = Average number of class i users in queue j. We denote by $\underline{N}$ the matrix

whose elements are N_{ij}.

T_{ij} = Average time spent by a class i transaction in queue j.

Once these are known, all the usual performance measures (response times, throughputs, utilizations) can be readily determined.

The mean value analysis uses three fundamental equations:

(1) Little's formula

$$N_{ij} = \lambda_i T_{ij} \qquad (i=1,2,\ldots,M;\ j=1,2,\ldots,K) \tag{1}$$

(2) Population balance

$$n_i = \sum_{j=1}^{K} N_{ij} \tag{2}$$

(3) Queueing discipline

$$T_{ij} = f_{ij}(\underline{N}^{(ij)}) \qquad (i=1,2,\ldots,M;\ j=1,2,\ldots,K) \tag{3}$$

where $\underline{N}^{(ij)}$ is the average conditional queue population at the time that a class $\overline{i}$ user arrives at queue j, and f_{ij} is a function whose form depends on the queueing discipline. Equation (3) expresses, in the most general form, any possible relation between queue populations and queueing times.

By summing (1) over j and substituting in (2) we find:

$$\lambda_i = n_i / \sum_k T_{ik} \tag{4}$$

so that, from (1):

$$N_{ij} = n_i T_{ij} / \sum_k T_{ik} \tag{5}$$

We now distinguish two cases:

(1) Finite population

It is reasonable to suppose that a class i user arriving at a queue would find there, on the average, the population that would have existed had the i-th user class contained n_i-1 (rather than n_i) users. Indeed, Sevcik and Mitrani [13] have proved that this is exactly so in many cases. Thus

$$N_{ij}^{(ij)}(n_1,\ldots,n_i,\ldots,n_M) = N_{ij}(n_1,\ldots,n_i-1,\ldots,n_M) \tag{6}$$

Clearly, $N_{ij}(0,0,\ldots,0) = 0$. Hence, starting at $n_i = 0$ (i=1,2...,M), we can apply equations (3) and (5) alternately with ever increasing values of the n_i, until the required population size is reached.

(2) Large populations (asymptotic case)

If the n_i are sufficiently large, the distinction between $\underline{N}$ and $\underline{N}^{(ij)}$ may be ignored, and (3) may be replaced with:

$$T_{ij} = f_{ij}(\underline{N}) \tag{7}$$

Equations (5) and (7) may now be solved simultaneously for $\underline{N}$ and $\underline{T}$ by means of the following iterative algorithm:

(1) Assign positive initial values to the N_{ij}.
(2) Compute the T_{ij}, using (7).
(3) Recompute the N_{ij}, using (5).
(4) Return to step (2).

The iterations are terminated when there is no significant change in the N_{ij} from one iteration to the next. An analogous scheme can be applied to the T_{ij}, using (5) first and then (7).

Let T^a_{ij} and N^a_{ij} be the solutions to (5) and (7). These are generally asymptotically valid, in the sense that if T_{ij} and N_{ij} are the true mean values, then $\lim(T^a_{ij}/N - T_{ij}/N)=0$ and $\lim(N^a_{ij}/N - N_{ij}/N)=0$, provided all n_i increase in constant proportions.

The iterations used in solving the asymptotic case require much less storage and CPU time than the recursions used in the finite case. However, simple heuristic corrections can be applied to the asymptotic formulas to generate very good approximations to the finite case [1,11,12].

(3) Processor sharing

With a processor sharing (PS) discipline, queueing time is obtained by multiplying service time by queue length. Hence, equation (3) takes the form

$$T_{ij} = (1+N_j^{(ij)})t_{ij} \tag{8}$$

where $N_j = \Sigma_i N_{ij}$ is the total average queue j population. Reiser and Lavenberg [1] show that using (6) and (8) recursively for finite populations yields exact results identical to those obtained from the product form solution.

In the asymptotic case, equation (7) becomes:

$$T_{ij} = N_j t_{ij} = t_{ij} \sum_k N_{kj} \tag{9}$$

Substituting (9) in (5) we obtain:

$$N_{ij} = n_i t_{ij} N_j / \sum_k t_{ik} N_k \tag{10}$$

Summing over i:

$$N_j = N_j \sum_i (n_i t_{ij} / \sum_k N_k t_{ik}) \tag{11}$$

Dividing both sides by N and substituting $x_j = N_j/N$ and $a_i = n_i/N$, we obtain:

$$x_j = x_j \sum_i (a_i t_{ij} / \sum_k x_k t_{ik}) \tag{12}$$

Note that x_j and a_i are the fractions of the total population in queue j and chain i, respectively. Equation (12) represents an iterative scheme for computing the x_j, starting out with any set of nonzero initial guesses. This scheme was derived and proven to converge by Pittel [2], and used by Bard [5,6] in a model of the VM/370 system. Modifications to equations (8)-(12) for queue-dependent service rates are straightforward.

With a little bit of algebra it is easily shown that (12) is equivalent to

$$x_j = x_j u_j \tag{12a}$$

where u_j is the utilization of the j-th queue, as given by equation (42). Equation (12a) can be interpreted as follows: Either the j-th queue is saturated with $u_j = 1$, in which case x_j can be positive. Or, the queue is not saturated, i.e. $u_j < 1$, in which case $x_j = 0$, i.e. the queue holds, on the average, a negligible fraction of the total population. See Section 9 for further discussion.

4. FCFS

The product form solution applies to queues with the first-come-first-serve (FCFS) discipline only if all user classes have exponential service time distributions with identical means. Mean value analysis permits relaxation of the latter restriction.

Let m_{ij} be the number of sojourns to queue j required by a class i transaction, and let θ_{ij} be the average service time per sojourn. we have, then:

$$t_{ij} = m_{ij}\theta_{ij} \qquad (i=1,2,\ldots,M;\ j=1,2,\ldots,K) \tag{13}$$

When a class i user enters queue j, he finds there on the average $N_{kj}^{(ij)}$ users of class k (k=1,2,..,M), with a total service time of $\Sigma_k N_{kj}^{(ij)}\theta_{kj}$. The duration of each sojourn will be $\theta_{ij} + \Sigma_k N_{kj}^{(ij)}\theta_{kj}$, for a total queueing time per transaction:

$$T_{ij} = m_{ij}(\theta_{ij} + \sum_k N_{kj}^{(ij)}\theta_{kj}) \tag{14}$$

This formula has been suggested by Reiser and Lavenberg [1]. Note that in the special case where θ_{ij} is the same for all user classes, equation (14) reduces to (8), showing that the solutions for PS and FCFS with equal exponential service times are identical.

In the asymptotic case, we substitute N_{kj} for $N_{kj}^{(ij)}$ and neglect θ_{ij} in comparison with $\Sigma_k N_{kj}\theta_{kj}$, so that

$$T_{ij} = m_{ij} \sum_k N_{kj}\theta_{kj} \tag{15}$$

Substituting in (5) we obtain:

$$N_{ij} = n_i m_{ij} A_j / \sum_k m_{ik} A_k \tag{16}$$

Where $A_j = \Sigma_i N_{ij}\theta_{ij}$. Equation (16) constitutes an iterative scheme for computing the N_{ij}. The amount of work can be reduced by multiplying by θ_{ij} and summing over i, which yields

$$A_j = A_j \sum_i (n_i t_{ij} / \sum_k m_{ik} A_k) \tag{17}$$

Equation (17) may be used to solve iteratively for the A_j, which may then be substituted in (16) to compute the N_{ij}. In analogy to (12a), equation (17) is equivalent to:

$$A_j = A_j u_j \tag{17a}$$

Example: Consider a network with two classes and three queues. Let

$$\underline{\theta} = \begin{bmatrix} 7 & 20 & 2 \\ 11 & 8 & 3 \end{bmatrix}$$

and

$$\underline{m} = \begin{bmatrix} 18 & 5 & 10 \\ 8 & 13 & 15 \end{bmatrix}$$

so that

$$\underline{t} = \begin{bmatrix} 126 & 100 & 20 \\ 88 & 104 & 45 \end{bmatrix}$$

A comparison between model and simulation results is presented in Table 1. It will be seen that the FCFS model gives acceptable results in both the finite and asymptotic cases, although convergence to the latter is fairly slow. In this problem, queues j=1,2 are saturated, whereas j=3 is not (see section 9).

5. Priorities

Mean value analysis does not lend itself easily to the treatment of ordinary priorities in FCFS queues. In mean value analysis, average queue lengths are substituted for the entire queue length distribution. Hence, it may (and in the asymptotic case it certainly will) appear as though the highest priority queue is never empty, so that lower priority users never receive service. Various standard tricks may be used: in case of preemptive priorities, for instance, the network may be solved as though only the highest priority user class is present. All servers are then assumed slowed down by a factor proportional to that class's utilization, the process is repeated for the second highest priority class, and so on. When priorities are not preemptive, it has been suggested by Schweitzer [11] that a term reflecting the expected remaining service time of the in-service user be added to the f_{ij} function in (3).

A type of priority scheduling that is particularly suitable for treatment by mean value analysis is based on service received. The rate of service received by a user in class i is to be proportional to his priority b_i. The rate of service is defined as some function (e.g. linear combination) of the service received at the various queues per unit of clock time.

Since each class i user completes λ_i/n_i transactions per time unit, his service rate may be expressed as:

$$R_i = (\lambda_i/n_i) \sum_j c_j t_{ij} = \left(\sum_j c_j t_{ij} \right) / \sum_j T_{ij} \qquad (18)$$

where the c_j are weights assigned to the various queues. For instance, in the VM/370 CPU-fair-share scheduler [7], c_j=1 for the CPU queue, and 0 for all others. In MVS, installations may specify arbitrary c_j for various resources [8].

It is required, then, that

$$R_i = b_i R \qquad (i=1,2,\ldots,M) \qquad (19)$$

where R is some unknown constant. Note, however, that (19) may not be satisfiable for user classes whose service demands are sufficiently small (e.g. I/O-bound users in a CPU-fair-share scheduler).

Assuming PS queues, we can control overall user service rates by assigning larger or smaller portions of a queue's service power to various users. Suppose, for example, that the j=1 queue can be controlled in this way. Let r_i be the fraction of processing power assigned to a class i user at that queue. Then we must have

$$r_i > 0 \quad (i=1,2,\ldots,M) \tag{20}$$

and, since the total processing power is unity:

$$\sum_i r_i N_{i1} = 1 \tag{21}$$

Furthermore, since at no time can more than full processing power be given to a single user, we must have:

$$r_i \leq 1 \quad (i=1,2,\ldots,M) \tag{22}$$

Note that (22) is not implied by (21), since we may have $N_{i1} < 1$ for some i.

We shall restrict ourselves to the asymptotic case. Equation (9) holds for all servers except j=1, where

$$T_{i1} = t_{i1}/r_i \tag{23}$$

Hence, $\Sigma_j T_{ij} = t_{i1}/r_i + \sum_{j\neq 1} t_{ij}N_j$. Using (18) and (19) we find:

$$t_{i1}/r_i + \sum_{j\neq 1} t_{ij}N_j = R_i/\sum_j c_j t_{ij} = b_i R/\sum_j c_j t_{ij} \tag{24}$$

and

$$r_i = \min\left[1,\ t_{i1}/(d_i R - \sum_{j\neq 1} t_{ij}N_j)\right] \tag{25}$$

where $d_i = b_i/\Sigma_j c_j t_{ij}$. Only values of R which make the denominator in (25) positive are admissible. From (21) we have:

$$\sum_i N_{i1}\min\left[1,\ t_{i1}/(d_i R - \sum_{j\neq 1} t_{ij}N_j)\right] = 1 \tag{26}$$

The following algorithm applies:

(1) Assign positive values to the N_{ij}.
(2) Find R so that (26) is satified.
(3) Compute the r_i from (25).
(4) Use (9) and (23) to compute the T_{ij}.
(5) Use (5) to recompute the N_{ij}.
(6) Return to step (2).

Note: If $\Sigma_i N_{i1} < 1$, Then ignore steps (2) and (3) and simply set all $r_i = 1$.

In the VM/370 model [5,6], CPU-fair-share scheduling with priorities was modeled via the admission policy to main storage. In cases where main storage was not a bottleneck, the fair share policy could not be modeled. The above algorithm can be used to overcome this difficulty.

6. Overlap

Classical queueing network analysis techniques require that no user be present in more than one queue at a time. In practice, however, many computer applications can overlap their own CPU and I/O activity. Such cases can be handled by means of mean value analysis.

For simplicity, assume that there are two queues whose services may be partly overlapped. Let t_{ij} be the non-overlappable service time, and t^*_{ij} the overlappable part ($j=1,2$), and let similar notation apply to N_{ij} and T_{ij}. We shall treat the asymptotic case with PS service. To equation (1) we must adjoin

$$N^*_{ij} = \lambda_i T^*_{ij} \tag{27}$$

Since users who are in the overlappable phase must be counted only once, equation (2) is replaced with:

$$n_i = \sum_j N_{ij} + \max_j N^*_{ij} \tag{28}$$

The approximation implied by (28) is that of using $\max_j E(N^*_{ij})$ as an estimate for $E(\max_j N^*_{ij})$.

The total number of users in queue j is $N_j+N^*_j$. Hence, equation (9) takes the form

$$T_{ij} = t_{ij} (N_j + N^*_j) \tag{29}$$

and,

$$T^*_{ij} = t^*_{ij} (N_j + N^*_j) \tag{30}$$

Trivial calculations result in:

$$\lambda_i = n_i/(\sum_k T_{ik} + \max_k T^*_{ik}) \tag{31}$$

so that

$$N_{ij} = n_i T_{ij}/(\sum_k T_{ik} + \max_k T^*_{ik}) \tag{32}$$

with an analogous equation for N^*_{ij}. Now, the total population in queue j is

$$N_j + N^*_j = \sum_i (N_{ij} + N^*_{ij}) =$$

$$= \sum_i [n_i(T_{ij} + T^*_{ij})/(\sum_k T_{ik} + \max_k T^*_{ik})] \tag{33}$$

so that (29) and (30) reduce to:

$$\begin{aligned} T_{ij} &= t_{ij} \sum_i [n_i(T_{ij} + T^*_{ij})/(\sum_k T_{ik} + \max_k T^*_{ik})] \\ T^*_{ij} &= t^*_{ij} \sum_i [n_i(T_{ij} + T^*_{ij})/(\sum_k T_{ik} + \max_k T^*_{ik})] \end{aligned} \tag{34}$$

Equations (34) define an iterative scheme for determining the T_{ij} and T^*_{ij}, and (32) can be used to compute N_{ij} and N^*_{ij}.

Examples:

1. Consider a case with two queues and two chains. The nonoverlappable service times are:

$$\underline{t} = \begin{bmatrix} 5 & 2 \\ 1 & 4 \end{bmatrix}$$

and the overlappable service times are:

$$\underline{t}^* = \begin{bmatrix} 2 & 3 \\ 3 & 2 \end{bmatrix}$$

The chain populations are $\underline{n} = (100,50)$. Computed and simulated response times are compared in Table 2. This is a case where only queue $j = 1$ is saturated.

2. When

$$\underline{t} = \begin{bmatrix} 5 & 3 \\ 1 & 5 \end{bmatrix}$$

$$\underline{t}^* = \begin{bmatrix} 2 & 2 \\ 3 & 4 \end{bmatrix}$$

both queues are saturated. The results for this case are shown in table 3. In both cases, model predictions are in good agreement with simulation results.

The above approach can be extended to overlaps among multiple queues, although the formulas can get quite complicated. A similar method can be used to model parallel processes with synchronization points, splitting and fusing processes, and other cases which arise in real operating systems.

7. Blocking

Blocking occurs when the progress of some transactions is impeded because of unavailability of resources held by other transactions.

Suppose the total amount of some resource (e.g. main storage) is S, and suppose S_i units are held by a class i transaction while, say, in queue j=2. Let j=1 refer to the queue of transactions waiting to be allocated resource before being admitted to queue j=2. No processing takes place while a transaction is in queue j=1. Again, we treat the asymptotic case with PS. Equations (1) and (2) hold as before, and so does (9) for all j>1. There are no explicit equations for T_{i1}. Instead, there is the resource constraint

$$\sum_i N_{i2} S_i \leq S \tag{35}$$

In addition, the sequence of admissions from queue j=1 to j=2 must be taken into account. If a user-class-independent admission policy (such as FCFS) is used, all waits to enter queue j=2 should be about the same, so that we can write

$$T_{i1} = T_1 m_{i2} \qquad (i=1,2,\ldots,M) \tag{36}$$

where T_1 is an unknown constant. Equation (5) takes the form:

$$N_{ij} = n_i T_{ij} / (T_1 m_{i2} + \sum_{k \neq 1} T_{ik}) \tag{37}$$

and (10) becomes:

$$N_{ij} = n_i t_{ij} N_j / (T_1 m_{i2} + \sum_{k \neq 1} t_{ik} N_k) \qquad (j=2,3,\ldots,K) \tag{38}$$

so that (35) may be rewritten as:

$$\sum_i [n_i t_{i2} N_2 S_i / (T_1 m_{i2} + \sum_{k \neq 1} t_{ik} N_k)] \leq S \tag{39}$$

The following algorithm may be applied:

1. Assign positive values to the N_{ij} $(i=1,2,\ldots,M;\ j=2,3\ldots,K)$.
2. Evaluate $S^* = \Sigma_i n_i t_{i2} N_2 S_i / (\Sigma_{k \neq 1} t_{ik} N_k)$. If $S^* \leq S$ the resource is not saturated and $T_1 = 0$. Otherwise, find T_1 (necessarily positive) to satisfy (39) with equality sign.
3. Recalculate the N_{ij} using (38).
4. Return to step (2).

This algorithm has been used successfully to model the scheduling policy of VM/370, with main storage as the constraining resource [5,6]. Trivial modifications of the algorithm can be used if the resource is held only during a specified fraction of a transaction's stay in queue j=2, or during stays in several different queues, or if there are several constrained resources. In using this algorithm, one implicitely makes the assumption that if the resource is not saturated on the average (i.e. its utilization is less than 1), then the average waiting time is negligible.

Let $u_s = S^*/S$ be the utilization of the blocked resource. Step 2 of the above algorithm ensures that either $u_s = 1$ or $T_1 = 0$. Shweitzer [11] has suggested that, in analogy to equation (12a), we may omit step 2, and instead use the formula $T_1 = T_1 u_s$ to update T_1 at each iteration.

8. Decomposable networks

Each queue in the network may itself have a complex internal structure. In particular, it may constitute a queueing network in its own right. In this case, the function $f_{ij}(\underline{N})$ is the average response time of the j-th subnetwork to a class i transaction, given that the population is N_{ij}. Evaluation of this function may itself require iterative procedures of the same kind as presented here for the overall network.

This approach is particularly useful when transitions within the subnetworks are much more frequent than between subnetworks, i.e. when the overall network is decomposable [9]. This method is used in the VM/370 model [5,6], where the overall network consists of an infinite-server terminal queue, a main storage queue (see Section 7 above), and a multiprogrammed-set queue. The latter, in turn, is a network of CPU and I/O queues. Solution proceeds by alternate iterations through the two levels of the network.

9. Response time calculation

Response time is defined as the time it takes a transaction to complete its service at all queues. Thus, when there is no overlap, we have

$$\tau_i = \sum_j T_{ij} \tag{40}$$

where τ_i is the average class i transaction response time. If there is overlap between services at different queues, then:

$$\tau_i = \sum_j T_{ij} + \max_j T^*_{ij} \tag{41}$$

In the finite population case, these formulas present no problems. In the asymptotic case, however, there are frequently queues for which the predicted populations N_{ij} and queueing times T_{ij} are zero. The reason is that strictly speaking, it is the fraction x_j of the total population in queue j that is being predicted, as in equation (12). As the total network population increases beyond bounds, it happens that some queues saturate: Their utilizations approach 100 percent, and their queue lengths grow beyond bound. Such queues have non-zero values of x_j. The remaining queues remain finite in length, thus accounting for a negligible portion of the total population. Hence, they have $x_j = 0$, and their utilization is below 100 percent.

When the network population is large, the terms corresponding to saturated queues dominate in (40), and the relative error in (40) or (41) is small even if $T_{ij}=0$ is used for the unsaturated queues. The true values of T_{ij} for unsaturated queues can, however, be estimated as follows: The saturated queues act as infinite Poisson sources for the unsaturated queues, so that the Pollaczek-Khinchine formula applies to the latter. Suppose queue j is an unsaturated single server queue with PS or FCFS with identical exponential service times for all classes. The utiliztion of that queue is

$$u_j = \sum_i \lambda_i t_{ij} \tag{42}$$

where the λ_i are calculated from (4), or from (1) applied to a saturated queue. We have, then:

$$T_{ij} = t_{ij}/(1 - u_j) \tag{43}$$

Other service distributions or disciplines, including preemptive or nonpreemptive priorities, can be handled by using the appropriate version of the Pollaczek-Khinchine formula (see, e.g. [10]).

10. Conclusion

Mean-value analysis, particularly in its asymptotic form, has been shown to be a versatile tool in the analysis of queueing networks. The method permits easy formulation and solution of many problems which do not have product form solutions. This is due, primarily, to the generality of the function f_{ij} appearing in equation (3), which makes it possible, for instance, to have the queueing time in one queue depend on conditions at other queues. The range of problems that can be treated is limited only by the ingenuity of the modeler. Most of the techniques given above already have direct practical applications within the VM/370 Model [5,6], and should help in expanding the range of systems for which good approximate analytical models can be constructed. Further work is required to establish error bounds on the approximate models, convergence rates

to the asymptotic formulas, and convergence proofs for some of the iterative algorithms.

References

(1) M. Reiser and S. Lavenberg, Mean Value Analysis of Closed Multiclass Queueing Networks, IBM Research Report RC7023, Yorktown Heights, N.Y. (1978).

(2) B. Pittel, Closed Exponential Networks of Queues with Blocking, the Jackson Type Stationary Distribution and its Asymptotic Analysis, IBM Research Report RC6174, Yorktown Heights, N.Y. (1976).

(3) J. P. Buzen, Fundamental Laws of Computer System Performance, Proceedings of the International Symposium on Computer Performance Modeling, Measurement, and Evaluation, Cambridge, Mass. (1976), pp. 200-210.

(4) H. Kobayashi and M. Reiser, On Generalization of Job Routing Behaviour in a Queueing Network Model, Research Report RC-5679, IBM T.J. Watson Research Laboratory, Yorktown Heights, N.Y. (1975).

(5) Y. Bard, the Modeling of some Scheduling Strategies for an Interactive Computer System, in Computer Performance, K.M. Chandy and M. Reiser (eds.), North Holland, Amsterdam (1977), pp. 113-138.

(6) Y. Bard, an Analytic Model of the VM/370 System, IBM J. Research and Development 22 (1978).

(7) IBM Virtual Machine Facility/370 System Extensions, General Information Manual, Form No. GC20-1827, IBM Data Processing Divison, white Plains, N.Y. (1977).

(8) H. W. Lynch and J.B. Page, the OS/VS2 Release 2 System Resources Manager, IBM Systems J. 13, 274-291 (1974).

(9) P. J. Courtois, 'Decomposability', Academic Press, N.Y. (1977).

(10) T. W. Gay and P.H. Seaman, Composite Priority Queue, IBM J. Research and Development 19, 78-81 (1975).

(11) P. Schweitzer, personal communication (1978).

(12) M. Reiser, Mean Value Analysis of Queuing Networks, a New Look at an Old Problem, IBM Research Report RC7228, Yorktown Heights, N.Y. (1978).

(13) K. C. Sevcik and I. Mitrani, The Distribution of Queueing Network States at Input and Output Instants, Research Report No. 307, IRIA, Rocquencourt (1978).

Chain populations	Transaction response times			
	Chain 1		Chain 2	
	Model	Simulation	Model	Simulation
5,5 (1)	1154	1140	1208	1156
50,50 (1)	12022	12267	9316	9070
100,100 (2)	26920	26120	16599	17136
150,150 (2)	40370	40740	24876	24627

Notes: (1) finite model
(2) asymptotic model

Table 1. Validation of FCFS model.

Chain	Transaction response times	
	Model	Simulation
1	1052	1013
2	604	638

Table 2. Validation of overlap model, example 1

Chain	Transaction response times	
	Model	Simulation
1	860	864
2	1075	1059

Table 3. Validation of overlap model, example 2

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

MEAN VALUE ANALYSIS OF QUEUING NETWORKS, A NEW LOOK AT AN OLD PROBLEM

M. Reiser

IBM T. J. Watson Research Center

Yorktown Heights, New York 10598

ABSTRACT

A new solution to queuing networks with product-form solution is given entirely in terms of mean queue size, mean waiting time and throughput. No need for normalization constants arises. The new analysis leads to simpler algorithms which have better numerical behavior than previous ones. It also is the basis for a heuristic method which for the first time allows solution of queuing systems with very many closed chains (>100). Such large systems arise in the context of communication system modeling.

1. INTRODUCTION

Queuing network theory has rapidly progressed since the fifties. The first thrust was by researchers in the field of Operations Research and culminated in J. R. Jackson's paper [1]. The work was picked up by Computer Scientists in the late sixties. More general classes of networks turned out to have product-form. The generalizations included multiple customer classes, queuing disciplines other than FIFO and generalized service time distributions. The most general class of such product-form networks is found in the important summary paper by F. Baskett, K. M. Chandy, R. R. Muntz and F. G. Palacios [2]. The technique used by all these researchers is to formulate the balance equations for a given class of networks, guess their solution (product-form) and verify its correctness by insertion into the equations. Chandy's local balance rule [3] served as a guide in guessing correctly.

Little attention, however, was paid to a strange property of the product-form solution, namely the fact that from the many parameters, necessary to specify a network much fewer entered into its solution. Take a closed exponential queuing network of the Jackson class, for example. Let P denote its $N \times N$ routing matrix (N is the number of queues). Instead of N^2 quantities p_{ij}, only N quantities θ_i given by the system of linear equations

$$\theta = \theta P \tag{1}$$

enter into the solution. Note that θ_i measures the *mean number of visits* a job makes to queue i between successive visits to a arbitrarily chosen queue i^*, for which $\theta_{i^*} = 1$ (note that (1) determines θ only up to a constant factor). Similarly, all the parameters of general phase-type distributions (when admissible) disappear from the product-forms of the queue-size distribution. All that matters is the *mean service time*. It was argued by H. Kobayashi and M. Reiser [4] that the crucial quantity in the product-form solution is the *mean work demand* brought into the system by jobs of a given class. How this work is divided into individual visits (as determined by the routing) was shown to be irrelevant. Statisticians call a system robust if only the mean enters into the solution. We find queuing networks with product-form solution remarkably robust with respect to routing and service-time distributions.

Such robustness asks for a simple physical explanation. We think that this paper gives such an explanation. Approaches to the queuing network problem different from the traditional algebraic method have indeed been tried recently. The operational method of P. Denning and J. Buzen is one example [5].

We shall describe the solution of queuing networks with product-form in terms of two intuitively appealing principles, namely

(1) Upon arrival, a customer sees the same closed system with himself removed (one less customer in the closed system) in long-term equilibrium;

(2) Little's law applied to the entire system and to each queue individually.

We feel that these two principles explain the fundamental mechanism governing such queuing systems. They lead to a much simpler evaluation procedure (than the previous convolution algorithms) and motivate heuristic methods for problems not solveable with current methods.

Before we give a detailed account of the mean value analysis, let us briefly comment on numerical methods. The product-form solution gives the joint queue-size distribution up to a normalization constant. This constant has a simple analytic expression in the case of an open queuing network but is a sum of product-terms in the case of closed systems. Because of the combinatorially growing state space, a naive summation is out of the question but for trivially small networks. J. P. Buzen published the first computationally efficient algorithm [6]. M. Reiser and H. Kobayashi independently discovered the same algorithm which they generalized to the multichain case [7]. In [8], they gave an interpretation of the algorithm in terms of convolutions. Their argument is that a closed system is equivalent to an associated open system conditioned to population size K. Then the normalization constant is simply the probablity that the open system contains exactly K customers, viz.

$$\text{Normalization-constant} = \Pr\{\text{population} = K\} = \Pr\{k_1+k_2+\ldots+k_N = K\} =$$

$$P_r\{k_1\} * Pr\{k_2\} * \ldots * Pr\{k_N\}, \Big|_{\text{at point } K}$$

Where k_i is the queue size of queue i, $Pr\{k_i\}$ is the queue size probability of queue i separated from the network and subjected to Poisson arrivals whose rate is θ_i as given by (1) and the asterisk denotes convolutions. M. Reiser [9] gives various efficient ways to calculate the convolutions and to evaluate

statistics such as mean queue size, throughputs and utilization factors. However, these convolutions are all time consuming and intermediate results may easily exceed the floating-point range of most computers even though final results are all reasonable in magnitude. The floating point range problem makes indeed infeasible the solution of perfectly well posed modeling problems (a large population of slow terminals, for example). The mean value analysis will not have any of these drawbacks.

2. CLOSED CYCLIC SYSTEM

The ideas behind the mean value analysis are easiest to describe for the example of a single closed chain as shown in Fig. 1. We denote by $S(K)$ the queuing system with K customers. For the moment, we consider a single class system only. Let

- K: Number of customers
- N: Number of queues
- τ_i: Mean service time of queue i,
- t_i: Mean waiting time of queue i (including service),
- n_i: Mean queue size of queue i (including customer in service),
- λ: Throughput of the chain.

We will use arguments if we want to emphasize that quantities are for the system with population size K, viz. $n_i(K)$, $t_i(K)$ and $\lambda(K)$ are mean queue size, mean waiting time and throughput of $S(K)$. For a memoryless system with FIFO queues we can straightforwardly write the equations

$$t_i(K) = \tau_i + \tau_i \times \{\text{mean number of customers seen upon arrival}\}, \tag{2}$$

$$\lambda(K) = K \Big/ \sum_{i=1}^{N} t_i(K), \tag{3}$$

$$n_i(K) = \lambda(K)\, t_i(K)\,. \tag{4}$$

Equation (3) states that an arriving customer in average has to wait for its own service time plus the backlog of work seen upon arrival. Equations (4) and (5) are simply Little's formula applied to the entire chain and to each queue respectively. Note that the average number of customers is of course K. If we had an expression for the bracketed term in (3) we could solve for all unknown quantities. Such an expression is found by means of

Theorem 1

In a closed queuing network with product-form solution, the probability to see state **k** upon customer arrival in $S(\mathbf{K})$ is the same as the long term equilibrium probability of **k** in $S(\mathbf{K}-1)$.

The proof of this theorem is found in [10]. However, equation (5) which result from the theorem can be proved more simply without the upon-arrival interpretation [11]. However, the physical significance of the theorem is more apparent in the given form.

From the theorem immediately follows the equation

$$t_i(K) = \tau_i + \tau_i\, n_i\,(K-1). \tag{5}$$

Note that (5) is not restricted to FIFO but applies for all rules consistent with product-form. Equation (5) together with (3) and (4) can be solved easily in a recursive manner, namely

$$n_i(0) = 0, \tag{6}$$

$$t_i(K) = \tau_i\,[1+n_i(K-1)], \tag{7}$$

$$\lambda(K) = K \,/\, \Sigma_i\, t_i(K), \qquad K>0,\ i=1,2,\ldots,N, \tag{8}$$

$$n_i(K) = \lambda(K)\, t_i(K). \tag{9}$$

3. GENERALIZATION TO THE FULL CLASS OF CLOSED PRODUCT-FORM SOLUTION NETWORKS

The recursion (6) to (9) generalizes easily to a network with general routing matrix P. Let i^* denote an arbitrarily chosen queue. Then, as mentioned before, the quantity θ_i uniquely defined by

$$\theta_{i^*} = 1, \tag{10}$$

$$\theta = \theta\, P \tag{11}$$

measures the average number of visits a customer makes to queue i between successive visits to the marked queue i^*. Since number of visits and throughput are proportional, θ_i also measures the throughput at queue i, λ_i, in units of λ_{i^*}, the throughput of queue i^*, viz.

$$\lambda_i = \theta_i \lambda_{i^*} , \; i=1,2,...N \tag{12}$$

If t_i measures the queuing time across queue i then obviously, the average time a customer needs between two successive departures from queue i^* is given by the sum of the products [average number of visits]×[mean waiting time], namely

$$\sum_{i=1}^{N} \theta_i t_i . \tag{13}$$

For simplicity of notation, define $\lambda^* = \lambda_{i^*}$. Now equations (7) to (9) translate into

$$t_i(K) = \tau_i [1+n_i(K-1)], \tag{14}$$

$$\lambda^*(K) = K / \Sigma_i \theta_i t_i(K), \tag{15}$$

$$n_i(K) = \lambda^*(K) \theta_i t_i(K) . \tag{16}$$

Define $t_i^* = \theta_i t_i$ and as usual traffic intensities $\rho_i = \theta_i \tau_i$, then we find

$$t_i^*(K) = \rho_i[1 + n_i(K - 1)], \tag{17}$$

$$\lambda^*(K) = K/\Sigma_i t_i^*(K), \tag{18}$$

$$n_i(K) = \lambda^*(K) t_i^*(K). \tag{19}$$

Equations (6) and (17) to (19) constitute the recursion to evaluate all quantities of $S(K)$ with general routing. They are of the same form as (7) to (9).

Next we shall discuss the multi-chain case. We assume that there are R disjoint closed routing chains. We will use superscripts r (r=1,2,...R) to denote that a given quantity belongs to chain r. We do not consider the case where customers change class membership within chains. Note however, that as shown in [7] this is only a trivial reduction of generality. Also we shall drop the asterisks used in (17) to (19). As before, we assume constant service rates. We introduce the notation

$\mathbf{K} = (K^1,K^2,...,K^R)$: population vector

$\mathbf{K}-\mathbf{e}_r = (K^1,K^2,...,K^{r-1},K^r-1,...,K^R)$: population vector with one less customer in chain r,

$R(i)$: Set of chains visiting queue i,

$Q(r)$: Set of queues in chain r,

λ^r: Throughput of marked queue in chain r,

τ_i^r: Mean service time of a chain r customer in queue i,

t_i^r: Mean time a chain r customer spends at queue i between successive visits to the marked queue of chain r,

n_i^r: Mean number of chain r customer at queue i,

$n_i = \Sigma_r\, n_i^r$: Mean queue size of queue i,

$\rho_i^r = \theta_i^r\, \tau_i^r$: Traffic intensity of chain r at queue i,

θ_i^r: Mean number of visits a chain r customer makes to queue i between successive visits to an arbitrarily marked queue in chain r.

Theorem 1 generalizes to the multiple chain case. In this case, the state upon arrivals of chain r customers has the same probability as the same state in a system with one less customer in chain r. However, it is much simpler to derive the following equations directly from the product-form solution as was done in [11]. We find

(1) The queuing discipline of queue i is processor sharing (PS) or preemptive-resume last-come, first-served (LCFS PR). The service time distribution may be general and have a different form for each chain. Then

$$t_i^r(\mathbf{K}) = \rho_i^r\,[1 + n_i^r\,(\mathbf{K}\text{-}\mathbf{e}_r)]. \tag{20}$$

Note that our definition of traffic intensity measures the mean time a customer spends at queue i between visits to the marked queue if there were no interference from other customers (congestion). The bracketed term in (20) is the *expansion factor* due to congestion, which we find simply related to the mean queue size.

(2) The queueing discipline is pure time delay, also called "infinite servers" (D). In this case, there is no expansion factor (by definition) and hence (20) becomes

$$t_i^r(\mathbf{K}) = \rho_i^r. \tag{21}$$

(3) The queuing discipline is first-come, first served (FCFS). The service-time distribution is exponential and does not depend upon customer's chain membership. The waiting-time equation is the same as (20). However, it is more instructive to write it in the form

$$t_i^r(\mathbf{K}) = \rho_i^r + \theta_i^r \Sigma_r \tau_i n_i^r (\mathbf{K}-\mathbf{e}_r). \tag{22}$$

The first term in (22) is the customer's own service demand, the sum measures the backlog of work he encounters upon arrival.

The recursion now follows analogous to (6) to (9) or (17) to (19) as follows

$$t_i^r(\mathbf{K}) = \begin{cases} \rho_i^r [1 + n_i(\mathbf{K}-\mathbf{e}_r)] & \text{cases 1 and 3} \\ \rho_i^r & \text{case 2,} \end{cases} \tag{23}$$

$$\lambda^r(\mathbf{K}) = K^r / \sum_{i \in Q(r)} t_i^r , \tag{24}$$

$$n_i(\mathbf{K}) = \sum_{j \in R(i)} \lambda^j(\mathbf{K}) t_i^j(\mathbf{K}) \tag{25}$$

where as usual $i=1,2,...N$, $r=1,2,...R$, $\mathbf{K} \geq 0$ and $n_i^r(\mathbf{0}) = 0$. Equation (23) to (25) allow for an easy recursive evaluation of the unknown quantities. Note that we do have to store all the intermediate values and that the loops over $\mathbf{K}$ should run in increasing order.

Finally, let us comment on the case of queue-dependent service rates which also lead to product-form solutions. Theorem 1 still holds but the mean waiting-time equations are more complicated than (20) to (22). They do involve terms of the marginal queue-size distribution of the queues with queue-dependent rates. The formulas as well as a very efficient way of calculating the marginal probabilities are given in [11]. We do not repeat them here for the sake of brefity.

4. NETWORKS WITH MANY CLOSED CHAINS

Closed chains find important applications in the modeling practice. Examples are

(1) To model job classes in a central server model [12],

(2) To represent application subsystems in a computing center model, and

(3) To represent flow controlled sessions in a communication network model [13].

In cases (1) and (2) the number of chains needed is often in the order of 10-25. In case (3) even more chains are required in a model of realistic size, e.g. 50-200.

The complexity of the recursion (23) to (25) for both storage and operation is of the order

$$\prod_{r=1}^{R} K^r . \tag{26}$$

Even on large computers, it will be impossible to solve for more than a small number of chains (less than 10 in most cases). Subsequently, we shall give a heuristic method derived from the mean value analysis, which will overcome the complexity barrier of (26) and which will allow analysis of problems with many chain (> 100 chains).

We note that the recursion is the source of the product in (26). It is our goal to replace this recursion by an iteration which is performed at the point **K** only.

Define quantities ϵ_i^r as follows

$$\epsilon_i^r(\mathbf{K}) = n_i(\mathbf{K}) - n_i(\mathbf{K}\text{-}\mathbf{e}_r). \tag{27}$$

Thus ϵ_i^r measures how the customer added to $S(\mathbf{K}\text{-}\mathbf{e}_r)$ is distributed over the individual queues. We have

$$\sum_{i=1}^{N} \epsilon_i^r = 1, \tag{28}$$

and also

$$0 \leq \epsilon_i^r \leq 1. \tag{29}$$

Suppose now that we had a function which would yield ϵ_i^r in terms of quantities of the system $S(\mathbf{K})$, for example

$$\epsilon_i^r = f\,(i, r, \{\lambda^j, j=1,2,...,R\}) \tag{30}$$

where we omitted the arguments denoting the chain population. We may now rewrite (23) to (25), again dropping the arguments (**K**), as follows

$$\epsilon_i^r = f\,(i, r, \{\lambda^j\}), \tag{31}$$

$$t_i^r = \rho_i^r [1 + n_i - \varepsilon_i^r], \tag{32}$$

$$\lambda^r = K^r / \sum_{i \in Q(r)} t_i^r, \tag{33}$$

$$n_i = \sum_{j \in R(i)} \lambda^j t_i^j . \tag{34}$$

We have now obtained a nonlinear system of equations which is independent of **K**, the source of the high operation count (26). We could solve (31) to (34) by a simple iteration method, starting with initial values for n_i and λ^r ($i=1,2,...N$, $r=1,2,...R$) and then iterating through (31) to (34) in a cyclic fashion until convergence is observed (or divergence established).

In its exact form, the function (30) is no less complex than the original problem. Clearly nothing is gained. It is our goal, hence, to obtain an efficient heuristic for (30). Our proposed heuristic will coalesce the R chains into a single chain which can be solved efficiently by the recursion (17) to (19). If a customer is removed from a chain, then all the values n_i^r $i=1,2,...N$; $r=1,2,...R$ are affected. However, since

$$\sum_{i \in Q(j)} n_i^j (\mathbf{K}) - n_i^j (\mathbf{K}-\mathbf{e}_r) = 0 \text{ for } j \neq r \tag{35}$$

(Where r is the chain with one less customer) we may assume that ε_i^r *is affected mostly by chain r.* Therefore, we estimate ε_i^r from a single chain problem with redefined parameters. The capacity of queue i's server devoted to chains $j=1,2,...R$, $j \neq r$ is given by

$$1 - \Sigma \lambda^j \tau_i^j \tag{36}$$

where the sum is over $j=1,2,...R$ but $j \neq r$. Taking the point of view of the fluid dynamic approximation of queuing systems, we may agree that a chain r customer "sees" a server with a reduced rate given by (36). Thus his mean service times are not τ_i^r but adjusted values

$$\hat{\tau}_i^r = \tau_i^r / (1+\lambda^r \tau_i^r - \sum_{j \in \mathbf{R}(i)} \lambda^j \tau_i^j). \tag{37}$$

Suppose that $\hat{n}_i^r(\mathbf{K})$ ($i=1,2,...N$) are the mean queue sizes of a single chain queuing problem with population K and with parameters given by (37). Then we set

$$\varepsilon_i^r = \hat{n}_i^r(\mathbf{K}^r) - \hat{n}_i^r(\mathbf{K}^r-1). \tag{38}$$

This completes our heuristic method.

The computational complexity of one iteration step through (31) to (34) is now

$$\sum_{r=1}^{R} K^r, \tag{39}$$

clearly an afordable effort even for large numbers of closed chains with seizable populations.

5. EXAMPLE AND ASYMPTOTIC PROPERTY

First we will give a numerical example of the heuristic method and compare it with exact results. We do not attempt to give a comprehensive empirical validation but shall argue that the approximation method is asymptotically valid as the population size increases.

The topology of our sample network is portrayed in Fig. 2. There are four chains (R=4) and eight queues (N=8). Mean service times and population sizes are listed in table 1.

Table 1: Parameters of the sample network

	Mean service times								Population
Queue	1	2	3	4	5	6	7	8	
Chain 1	2	2	2	–	2	–	–	–	6
Chain 2	–	0.5	0.5	0.5	–	2	–	–	8
Chain 3	4	4	–	–	–	–	3		4
Chain 4	1	–	–	1	–	–	–	5	8

The example is chosen to resemble a communication network model. Queues 1 to 4 represent half-duplex links whereas queues 5 to 8 model sources. Each chain is a virtual channel with a flow control window of size K^r, r=1,2,...4 (for a detailed description of the communication network model see [13]). Results for mean delay time and for throughput are listed in table 2. The mean delay is defined from leaving the sources until arriving at the destination.

Table 2: Results for the network of Fig. 1.

Chain	1	2	3	4	
	28.99	5.72	47.20	7.65	exact
Mean delay	29.25	5.61	48.50	7.58	iterative
	0.89	2.03	2.75	0.97	%error
	0.188	0.491	0.0786	0.200	exact
Throughput	0.186	0.478	0.0765	0.198	iterative
	1.08	2.72	2.75	1.01	%error

We find the errors quite small, clearly adequate for any practical purpose. Similar observation was made for a central server model with four chains (errors are less than 5%).

The convergence of the iterative method was found to be uncritical and rapid. The difference between successive iteration steps (measured in terms of a suitable norm) decrease exponentially (i.e. the scheme has linear convergence property) as would be expected for a simple first order iteration. The initial condition was found entirely uncritical.

We do not have currently proof that the scheme converges nor do we know error bounds. There is an important limiting case, however, where convergence and accuracy was proved namely

$$\mathbf{K} \to \infty \text{ such that } \mathbf{K}^r/\mathbf{K} = \alpha^r = \text{const.} \tag{40}$$

where $|\mathbf{K}| = \Sigma_r K^r$. Define $\nu_i = n_i/|\mathbf{K}|, \mu_i^r = t_i^r/|\mathbf{K}|$ and $K = |\mathbf{K}|$. Then we may rewrite (32) to (34) as follows

$$\mu_i^r = \rho_i^r [(1-\epsilon_i^r)/K + \nu_i] \tag{41}$$

$$= \rho_i^r \nu_i + \mathbf{O}(1/K),$$

$$\lambda^r = \alpha^r / \sum_{i \in Q(r)} \mu_i^r, \tag{42}$$

$$\nu_i = \sum_{j \in R(i)} \lambda^j \mu_i^j . \tag{43}$$

B. Pittel [14] proved that the term $\mathbf{O}(1/K)$ in (41) can be neglected and that the resulting iteration describes the limiting case (40). He also proved convergence of the iterative scheme (41) to (43). This result gives us a very good reason to trust the heuristic scheme. Note that the iteration (41) to (43) is a

means to locate the *bottlenecks* of the queuing system. Bottleneck queues are identified by $v_i=1$. Unlike in the case of an open queuing problem or a closed single chain problem, the *bottlenecks of a closed multichain problem are not found by inspection of the parameters.* Throughput values follow from the knowledge of bottlenecks. From this discussion, we expect the iteration (31) to (34) to be the more accurate, the larger the population and the more closed chains there are in the network. Also, throughputs are more trustworthy than mean queue size and mean delays. These properties have in fact been observed from the examples which we ran.

6. CONCLUSION

We have given an analysis of queuing networks solely based on theorem 1 (which gives the system state upon customer arrival) and Little's formula. Both, theorem 1 and Little's formula have a simple, physically meaningful interpretation. Our analysis involves only the most widely used statistics such as mean queue size, mean delay, throughput and utilization. No joint distribution of product-form or normalization constants are involved. The mean value analysis leads to simple recursive algorithms. Even though the computational complexity is the same as for the convolution algorithm in its most efficient form, the mean value analysis avoids problems of floating point overflow/underflow inherent in the earlier algorithms. Also it is much simpler to program.

Our opinion that the mean value analysis reveals a deep property of the solution in physically meaningful terms is substantiated by the fact, that it led us to a heuristic algorithm which overcomes the complexity barrier of the exact recursion and allows the solution of problems with many closed chains (>100). Such problems arise in the context of communications networks and computer models with a large number of application classes. In those case which we compared to exact results, the heuristic is accurate to a few percents which is clearly adequate in practice. We have also shown that it is asymptotically valid as the population size increases. The heuristic also leads us into the area of networks without product-form solution. Generalizations for FCFS queues with class-dependent, non-exponential service time distributions are found in [13].

REFERENCES

[1] J. R. Jackson, "Jobshop-Like Queuing Systems," Management Sci., 10, October 1963, pp. 131-142.

[2] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios, "Open, Closed, and Mixed Networks of Queues with Different Classes of Customer," JACM, 22, April 1975, pp. 248-260.

[3] K. M. Chandy, "The Analysis and Solution for General Queuing Networks", Proc. 7th Annual Princeton Conference on Information Sciences and Systems, Princeton University, 1973, pp. 428-434.

[4] H. Kobayashi and M. Reiser, "On Generalization of Job Routing Behavior in a Queuing Network Model", IBM Research Report RC 5252, Yorktown Heights, 1975.

[5] J. P. Buzen, "Operational Method".

[6] J. P. Buzen, "Computational Algorithms for Closed Queuing Networks with Exponential Servers," CACM, 16, September 1973, pp. 527-531.

[7] M. Reiser and H. Kobayashi, "Queuing Networks with Multiple Closed Chains: Theory and Computational Algorithms," IBM J. Res. and Develop., 19, May 1975, pp. 283-294.

[8] M. Reiser and H. Kobayashi, "On the Convolution Algorithm for Separable Queuing Networks", Proc. Intl. Symp. Computer Performance Modeling, Measurement and Evolution, Cambridge Massachusetts, March 29-31, 1976, pp. 109-117.

[9] M. Reiser, "Numerical Methods in separable Queuing Networks," Studies in the Management Sciences, vol. 7, 113-142 (1977).

[10] S. S. Lavenberg and M. Reiser, "The State Seen by an Arriving Customer in Closed Multiple Chain Queuing Networks," to appear.

[11] M. Resier and S. S. Lavenberg, "Mean Value Analysis of Closed Multichain Queuing Networks," IBM Research Report RC 70 23, Yorktown Heights, N.Y., 1978.

[12] Y. Bard, "The Modeling of Some Scheduling Strategies for an Interactive Computer System," in Computer Performance, K. M. Chandy and M. Reiser, Editors, North Holland Publishing Co., 1977, pp. 113-137.

[13] M. Reiser, "A Queuing Network Analysis of Computer Communication Networks with Window Flow Control", IBM Research Report RC 7218, Yorktown Heights, N¬Y¬, 1978.

[14] B. Pittel, "Closed Exponential Networks of Queues, Asymptotic Analysis," IBM Research Report RC 6174, Yorktown Heights, N.Y., 1976.

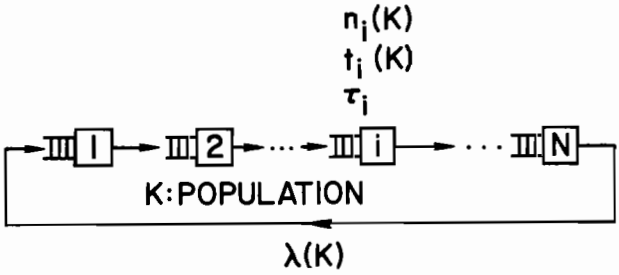


Fig. 1. A simple closed cyclic chain

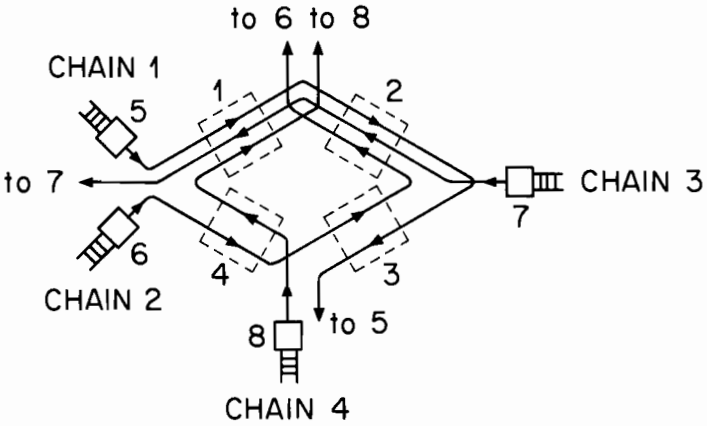


Fig. 2. Example of a closed multichain Network with four chains (R=4) and eight queues (N=8).

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

A COMPUTATIONAL ALGORITHM FOR QUEUE DISTRIBUTIONS VIA THE PÓLYA THEORY OF ENUMERATION

Hisashi Kobayashi
Computer Sciences Department
IBM Thomas J. Watson Research Center
Yorktown Heights, New York 10598

We present a new computational algorithm for evaluating the queue distribution in a general Markovian queuing network, based on the Pólya theory of counting. We formulate queue size vectors as equivalence classes relative to a symmetric group. The normalization constant of the queue-distribution then corresponds to the pattern inventory in the Pólya theory. A central server model is discussed as an application example of this new algorithm.

I. INTRODUCTION

A "network of queues" representation provides a basic framework in dealing with the performance analysis of multiple resource systems, in which different resources process jobs asynchronously to each other. The class of models for which we find a simple closed solution of the equilibrium queue distribution is the so-called "Markovian queuing network" [1-4]. For this class the equilibrium distribution is given in "product" form. This expression, however, includes a normalization constant, and determination of the normalization constant presents a computationally nontrivial task.

A number of authors have proposed various algorithms designed to evaluate efficiently the normalization constant, and related performance measures - utilization, throughput, moments of queue size, average response time, etc. In the present paper we propose a new algorithm that is derived based on the Pólya theory of enumeration - a well-discussed subject in books on combinatorial mathematics [5-8]. The Pólya theory of enumeration influenced the research in finding minimal cost networks for the realization of switching functions, as treated by Slepian [9] and Harrison [10]. The problem of evaluating the normalization factor of queue distribution is a bona fide combinatorial problem, thus it is quite natural to investigate possible applications of the Pólya theory to queuing theory.

II. STATEMENTS OF THE PROBLEM

Consider a closed* queuing network which consists of M service stations arbitrarily connected to each other. Let us define the following set of nomenclature concerning the analysis of such network:

$$M = \{1,2,3,\ldots,M\}: \quad \text{the set of service stations} \qquad (2.1)$$

$$N = \sum_{i \in M} n_i: \quad \text{the network population} \qquad (2.2)$$

*In the original paper [14] a more general class of queuing networks is discussed.

$$F(N) = \{\underline{n} | n_i > 0 \text{ for all } i \epsilon M \text{ and } \sum_{i \epsilon M} n_i = N\}: \quad \text{the set of feasible queue vectors} \tag{2.3}$$

$C_i(n)$ = the processing rate of server i, when its local queue size is n, $i \epsilon M$ (2.4)

$$\beta_i(n) = \prod_{k=1}^{n} \frac{1}{C_i(k)}, \quad i \epsilon M \tag{2.5}$$

W_i = the expected total work (or service) a job demands from server i during this job's entire life in the network. (2.6)

In order to obtain the equilibrium state distribution of the queue-size vector $p(\underline{n})$, we make a set of fairly general assumptions (see [2,3,4] for details) concerning (i) the routing behavior, (ii) service (or work) distribution, (iii) service (or processing) rates, and (iv) queue disciplines. We can then obtain the following product form solution:

$$p(\underline{n}) = \begin{cases} c \prod_{i \epsilon M} f_i(n_i), & \text{if } \underline{n} \epsilon F(N) \\ 0, & \text{if } \underline{n} \notin F(N) \end{cases} \tag{2.7}$$

where the functions $f_i(n_i)$ are themselves given in the following product form:

$$f_i(n_i) = \beta_i(n_i) W_i^{n_i}, \quad i \epsilon M \tag{2.8}$$

the scalar constant c of Equation (2.7) is the normalization factor referred to in Section I and is given by

$$c = 1/g(M,N) \tag{2.9}$$

where

$$g(M,N) = \sum_{\underline{n} \epsilon F(N)} \prod_{i \epsilon M} \beta_i(n_i) W_i^{N_i} \tag{2.10}$$

thus the problem is reduced to that of evaluating $g(M,N)$ for a given pair (M,N).

The convolutional algorithm of Buzen [11] and Reiser and Kobayashi [12,13] is essentially the following recursive formula:

$$g(M,N) = \sum_{k=0}^{N} g(M-1,N-k) \beta_M(k) W_M^k, \quad M \geq 1, \ N \geq 1 \tag{2.11}$$

with the boundary conditions

$$g(M,0) = 1, \quad \text{for } M \geq 0, \tag{2.12a}$$

and

$$g(0,N) = \begin{cases} 1, & \text{for } N=0, \\ 0, & \text{for } N\geq 1. \end{cases} \tag{2.12b}$$

for a fixed value of M, the sequence $\{g(M,i);\ 0\leq i\leq N\}$ is the convolutional sum of the sequence $\{\beta_M(i)W_M^i;\ 0\leq i\leq N\}$ and $\{g(M-1,i);\ 0\leq i\leq N\}$. The computation of $\{g(M,i);\ 0\leq i\leq N\}$ given the value of $\{g(M-1,i);\ 0\leq i\leq N\}$ requires $\frac{N(N+1)}{2}$ multiplications and additions. Thus, for a given value of the pair (M,N) the evaluation of $g(M,N)$ requires, in total, $\frac{(M-1)}{2}\sum_{n'=1}^{N} n'(n'+1) = \frac{(M-1)N(N+1)(N+2)}{6}$ multiplications and additions.

Under the special condition of constant service rates of the form for all $i\epsilon M$:

$$C_i(n) = \begin{cases} C_i & \text{for } n\geq 1 \\ 0 & \text{for } n=0 \end{cases} \tag{2.13}$$

we find the following simple recurrence algorithms for the two-dimensional array $\{g(M,N)\}$:

$$g(M,N) = g(M-1,N) + \tau_M g(M,N-1), \quad M\geq 1,\ N\geq 1 \tag{2.14}$$

with the boundary conditions (2.12). The parameter τ_i is the expected <u>total</u> service time given to a job by server i during the job's lifetime within the network, and is given by

$$\tau_i = \frac{W_i}{C_i}, \quad i\epsilon M \tag{2.15}$$

The evaluation of $g(M,N)$ requires, for this special case, $(M-1)N$ multiplications and additions.

III. <u>A NEW COMPUTATIONAL ALGORITHM</u>

We now introduce a new algorithm for evaluating the normalization constant $g(M,N)$. This algorithm is restricted to a network with exponential servers all of which have fixed service rates, i.e., the case where Equation (2.13) is true for all $i\epsilon M$. Then certainly we could use the recursive formula (2.14) throughout the entire steps, starting with the boundary condition (2.12). The evaluation of $\{g(m,n);\ 1\leq n\leq N,\ 1\leq m\leq M\}$ would require only $(M-1)N$ multiplications and additions. However, the computational formula to be discussed below is sometimes more convenient, especially when N is small.

The assumption of the constant service rates of (2,13) allows us to write $g(M,N)$ of (2.10) as

$$g(M,N) = \sum_{\underline{n}\epsilon F(N)} \prod_{i\epsilon M} \tau_i^{n_i} \tag{3.1}$$

where τ_i was defined by (2.15). Let us define the set of stations

$$M = \{1,2,\ldots,M\} \tag{3.2}$$

and the set of N jobs

$$N = \{1,2,\ldots,N\} \tag{3.3}$$

Consider then a set of functions that have N and M as their domain and range, respectively:

$$F = \{f \mid f:N \to M\} \tag{3.4}$$

A function f in the set F represents a way of placing N jobs into M service stations. We write, for example,

$$f(j) = i, \quad j \in N, \quad i \in M \tag{3.5}$$

which implies that job j is placed in station i.

Consider a permutation π defined over N, and let S_N be the set of all permutations defined over N:

$$S_N = \{\pi \mid \pi:N \to N\} \tag{3.6}$$

The elements of S_N form a symmetric group of degree N. For a given function $f_1 \in F$ and permutation $\pi \in S_N$, we can define another function f_2 by

$$f_2(j) = f_1(\pi(j)), \quad j \in N \tag{3.7}$$

Clearly the function f_2 is also a member of F. However, such functions f_1 and f_2 correspond to the same queue size vector $\underline{n}$

$$\underline{n} = [n_1, n_2, \ldots, n_M] \tag{3.8}$$

since we do not distinguish the individual jobs. Therefore, we say that the functions f_1 and f_2 are equivalent relative to the permutation group S_N. Distinct values of $\underline{n} \in F(N)$ correspond to distinct equivalence classes.

We interpret the parameter τ_i of (2.15) as the weight of element i in the set M, and thus

$$\sum_{i \in M} \tau_i \tag{3.9}$$

represents the inventory of the set M. If a function f belongs to the equivalence class $\underline{n}$ (3.8), then the weight $W(f)$ of the function f is

$$W(f) = \prod_{i \in M} \tau_i^{n_i}, \quad \text{for all } f \in \underline{n} \tag{3.10}$$

which is called the weight of the equivalence class $\underline{n}$. Then the pattern inventory of F - the sum of weights of distinct equivalence classes relative to the permutation group S_N - is

$$\sum_{\underline{n} \in F(N)} W(f) \tag{3.11}$$

which is nothing but $g(M,N)$ of (3.1)! This observation immediately calls our attention to the celebrated Pólya theorem:

Theorem (Pólya): The pattern inventory $g(M,N)$ of the set of the equivalence classes of functions from the domain N to the range M is

$$g(M,N) = Z_{S_N}\left(\sum_{i\in M} \tau_i, \sum_{i\in M} \tau_i^2, \ldots, \sum_{i\in M} \tau_i^N\right) \tag{3.12}$$

where $Z_{S_N}(x_1,x_2,\ldots,x_N)$ is the cyclic index polynomial of the permutation group S_N.

The cycle index polynomial of S_N is given from Cauchy's formula

$$Z_{S_N}(x_1,x_2,\ldots,x_N) = \sum \frac{x_1^{\mu_1} x_2^{\mu_2} \cdots x_N^{\mu_N}}{\mu_1!\, 2^{\mu_2} \mu_2! \cdots N^{\mu_N} \mu_N!} \tag{3.13}$$

where the sum is taken over the set of distinct M tuples, $\langle \mu_i;\ i = 1,2,\ldots,M\rangle$ such that

$$\sum_{i\in M} i\mu_i = N \tag{3.14}$$

Table 1 tabulates (3.13) for $N = 1,2,\ldots,7$.

Table 1

Cycle Index Polynomials of Symmetric Groups

N	Z_{S_N}
1	x_1
2	$1/2(x_1^2 + x_2)$
3	$1/6(x_1^3 + 3x_1x_2 + 2x_3)$
4	$1/24(x_1^4 + 6x_1^2x_2 + 3x_2^2 + 8x_1x_3 + 6x_4)$
5	$1/120(x_1^5 + 10x_1^3x_2 + 15x_1x_2^2 + 20x_1^2x_3 + 20x_2x_3 + 30x_1x_4 + 24x_5)$
6	$1/720(x_1^6 + 15x_1^4x_2 + 45x_1^2x_2^2 + 15x_2^3 + 40x_1^3x_3 + 120x_1x_2x_3 + 40x_3^2 + 90x_1^2x_4 + 90x_2x_4 + 144x_1x_5 + 120x_6)$
7	$1/5040(x_1^7 + 21x_1^5x_2 + 70x_1^4x_3 + 105x_1^3x_2^2 + 210x_1^3x_4 + 420x_1^2x_2x_3 + 105x_1x_2^3 + 280x_1x_3^2 + 630x_1x_2x_3 + 504x_1^2x_5 + 840x_1x_6 + 210x_2^2x_3 + 504x_2x_5 + 420x_3x_4 + 720x_7)$

Thus all that is required is to compute the set of values

$$x_k = \sum_{i \in M} \tau_i^k, \quad k = 1,2,\ldots \tag{3.15}$$

and substitute them into the polynomial Z_{S_N}.

Alternatively, we recursively compute $g(m,n)$, $1 \le n \le N$, $1 \le m \le N$. We can derive the following expression for the cycle index polynomials:

$$Z_{S_n}(x_1,x_2,\ldots,x_n) = \begin{cases} \frac{1}{n} \sum_{k=0}^{n-1} x_{n-k} Z_{S_k}(x_1,x_2,\ldots,x_k), & \text{for } n \ge 1 \\ 1, & \text{for } n=1 \end{cases} \tag{3.16}$$

which leads to the recurrence relation of the sequence $g(M,n)$, $n = 1,2,3,\ldots$

$$g(M,n) = \frac{1}{n} \sum_{k=0}^{n-1} x_{n-k} g(M,k)$$

$$= \frac{1}{n} \sum_{k=1}^{n} x_k g(M,n-k) \tag{3.17}$$

with the initial condition

$$g(M,0) = 1, \quad M \ge 1 \tag{3.18}$$

Note that Equation (3.17) is also of a convolutional form: we can view the sequence $\{g(M,n): n = 1,2,\ldots\}$ as an autoregressive sequence with varying regressive coefficients $\{\frac{1}{n} x_{n-k}: k = 0, 1, \ldots, n-1\}$.

IV. AN APPLICATION EXAMPLE

The computational formulas presented above will be of practical interest when there are many servers in the network. The cost of computing the parameters $\{x_k, k = 1,2,\ldots\}$ of (3.15) is insignificant in many cases of practical interest. Consider, for example, a central server model in which the CPU station is followed by a number of I/O devices (disks and drums) with a number of independent access paths in parallel: if the traffic distribution to different paths is uniform (which is often assumed in the absence of detailed measurement data), then the model becomes a closed network with many independent servers, but with the same parameter value of $\{\tau_i\}$.

For example, a model of an interactive system with multiprogramming in virtual storage can be decomposed into the outer model - a time-shared system model - and the inner model - a central server model [2,3]. Figure 1 shows a typical structure of the inner model with M=16: servers 1 through 10 represent magnetic drum sectors with independent access paths; servers 11 through 15 are magnetic disks with independent channels; and server 16 represents CPU. The multiprogramming level, N, varies as time changes. Usually the value N is controlled through the job scheduler. We assume the following workload parameters per interaction, where an interaction starts when an interactive user creates a

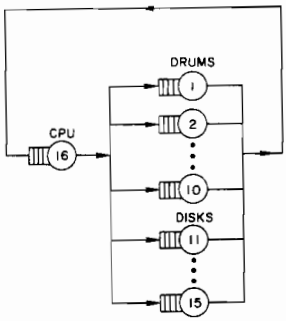


Figure 1. A Closed Queuing Network
Model With M=16 Stations

request (or job) and it ends when the job is processed by the system and its response is received by the user.

Average CPU work per interaction:	$W_{16} = 2.0$ sec.
Average number of drum accesses (reads) per interaction:	$R_{drm} = 80$
Average number of disk accesses (reads) per interaction:	$R_{dsk} = 20$
Average latency and transfer time per drum access:	20 msec.
Average seek, latency and transfer time per disk access:	100 msec.

In the absence of measurement data concerning how these drum reads and disk reads are distributed among the separate access paths, we assume the uniform distributions:

$$W_1 = \ldots = W_{10} = 20 \text{ msec} \times \frac{R_{drm}}{10} = 0.16 \text{ sec};$$

$$W_{11} = \ldots = W_{15} = 100 \text{ msec} \times \frac{R_{dsk}}{5} = 0.40 \text{ sec}.$$

Since the service (or work) is represented in time, the processing rate $\{C_i\}$ should be set to unity. Hence the parameters $\{\tau_i\}$ of (2.15) are the same as $\{W_i\}$:

$$\tau_1 = \ldots = \tau_{10} = 0.16 \text{ sec};$$

$$\tau_{11} = \ldots = \tau_{15} = 0.40 \text{ sec};$$

$$\tau_{16} = 2.0 \text{ sec}.$$

We first compute the parameters x_i's of (3.15)

$$x_1 = 10 \times 0.16 + 5 \times 0.4 + 2.0 = 5.6 \text{ sec};$$

$$x_2 = 10 \times 0.16^2 + 5 \times 0.4^2 + 2.0^2 = 5.056 \text{ sec}^2;$$

$$x_3 = 10 \times 0.16^3 + 5 \times 0.4^3 + 2.0^3 = 8.361 \text{ sec}^3;$$

$$x_4 = 10 \times 0.16^4 + 5 \times 0.4^4 + 2.0^4 = 16.135 \text{ sec}^4,$$

etc. Then from Formula (3.12) and the polynomials of Table 1 (alternatively from the recurrence formula (3.17)), we obtain

$$g(16,0) = 1$$

$$g(16,1) = x_1 = 5.6 \text{ sec};$$

$$g(16,2) = \frac{1}{2}(x_1^2 + x_2) = 18.2 \text{ sec}^2;$$

$$g(16,3) = \frac{1}{6}(x_1^3 + 3x_1x_2 + 2x_3) = 46.2 \text{ sec}^3;$$

$$g(16,4) = \frac{1}{24}(x_1^4 + 6x_1^2x_2 + 3x_2^2 + 8x_1x_3 + 6x_4) = 103.4 \text{ sec}^4;$$

etc. Utilization $\rho_i(N)$ of server i for the degree of multiprogramming N is given (see e.g., [2]) by

$$\rho_i(N) = W_i \frac{g(M,N-1)}{g(M,N)} \tag{4.1}$$

We can predict, for example, CPU utilization under different values of multiprogramming level, N, as follows:

$$\rho_{16}(1) = \frac{2.0}{5.6} = 0.36;$$

$$\rho_{16}(2) = 2.0 \times \frac{5.6}{18.2} = 0.62;$$

$$\rho_{16}(3) = 2.0 \times \frac{18.2}{46.2} = 0.79;$$

$$\rho_{16}(4) = 2.0 \times \frac{46.2}{103.4} = 0.89;$$

An alternative formula for utilization $\rho_M(N)$ for the M^{th} resource is given from Equations (2.14) and (4.1) as

$$\rho_M(N) = 1 - \frac{g(M-1,N)}{g(M,N)} \tag{4.2}$$

For the degree of multiprogramming $N=4$, for example, we need to calculate $g(15,4)$. For this purpose we compute the following parameters:

$$y_1 = 10 \times 0.16 + 5 \times 0.4 = 3.6 \text{ sec}$$

$$y_2 = 10 \times 0.16^2 + 5 \times 0.4^2 = 1.056 \text{ sec}^2$$

$$y_3 = 10 \times 0.16^3 + 5 \times 0.4^3 = 0.361 \text{ sec}^3$$

$$y_4 = 10 \times 0.16^4 + 5 \times 0.4^4 = 0.135 \text{ sec}^4$$

Then

$$g(15,4) = \frac{1}{24}(y_1^4 + 6y_1^2y_2 + 3y_2^2 + 8y_1y_3 + 6y_4) = 11.0.$$

Hence

$$\rho_{16}(4) = 1 - \frac{11.0}{103.4} = 0.89$$

which is, not surprisingly, the same as the value obtained earlier.

The k^{th} moment of the number of customers, n_i, is given [2] by

$$E[n_i^k] = \frac{1}{g(M,N)} \sum_{n=1}^{N} g(M,N-n)[n^k-(n-1)^k]\tau_i^n \qquad (4.3)$$

For instance, the average of CPU queue for the degree of multiprogramming N=4 is

$$E[n_{16}] = \frac{1}{g(16,4)} \sum_{n=1}^{4} g(16,4-n)2.0^n$$

$$= \frac{1}{103.4}(46.2 \times 2.0 + 18.2 \times 2.0^2 + 5.6 \times 2.0^3 + 1 \times 2.0^4$$

$$= 2.19$$

Similarly, we obtain the average queue sizes the the drums and disks:

$$E[n_1] = \ldots = E[n_{10}] = 0.076$$

$$E[n_{11}] = \ldots = E[n_{15}] = 0.210$$

We check that these values add up to N=4:

$$2.19 + 0.76 \times 10 + 0.210 \times 5 = 4.0$$

References

|1| L. Kleinrock (1975). Queueing Systems, Vol. I: Theory, John Wiley & Sons, New York.

|2| H. Kobayashi (1978). Modeling and Analysis: An Introduction to System Performance Evaluation Methodology, Addison-Wesley, Reading, Mass.

|3| H. Kobayashi (1978). "System Design and Performance Analysis Using Analytic Models" in Current Trends in Programming Methodology Vol. III: Software Modelling, (Ed. M. Chandy and R.T. Yeh), pp. 72-114, Prentice-Hall Inc., Englewood Cliffs, N.Y.

|4| H. Kobayashi and A.G. Konheim (1977). "Queuing Models for Computer Communications System Analysis" (Invited paper). IEEE Trans. on Communications, COM-25, No. 1, pp. 2-29.

|5| C.L. Liu (1968). Introduction to Combinatorial Mathematics, McGraw-Hill Book Co., New York.

|6| C. Berge (1971). Principle of Combinatorial Mathematics. Academic Press, New York.

|7| H.S. Stone (1973). Discrete Mathematical Structures and Their Applications, Science Research Associate, Inc., Chicago.

|8| F.P. Preparata and R.T. Yeh (1973). Introduction to Discrete Structures for Computer Science and Engineering, Addison-Wesley, Reading, Mass.

|9| D. Slepian (1953). "On the Number of Symmetry Types of Boolean Functions of n Variables", Can. J. Math., Vol. 5, No. 2, pp. 185-193.

|10| M.H. Harrison (1965). Introduction to Switching and Automata Theory, McGraw-Hill Book Co., New York.

|11| J.P. Buzen (1973). "Computational Algorithms for Closed Queuing Networks with Exponential Servers", Comm. of ACM, Vol. 16, No. 9, pp. 527-531.

|12| M. Reiser and H. Kobayashi (1973). "Recursive Algorithms for General Queuing Networks with Exponential Servers", IBM Research Report, RC-4254, IBM Research Center, Yorktown Heights, N.Y.

|13| M. Reiser and H. Kobayashi (1975). "Queuing Networks with Multiple Closed Chains: Theory of Computational Algorithms", IBM J. of Res. and Develop., Vol. 19, No. 3, pp. 283-294.

|14| H. Kobayashi (1976). "A Computational Algorithm for Queue Distribution via Pólya Theory of Enumeration", IBM Research Report, RC-6154, IBM Research Center, Yorktown Heights, N.Y.

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

A DIRECT NUMERICAL METHOD FOR QUEUEING NETWORKS

William J. Stewart
Department of Computer Science
North Carolina State University
Raleigh, North Carolina 27650
U.S.A.

A direct numerical method for the solution of queueing networks is presented. The problem of matrix "fill-in" usually associated with such methods is discussed and a fixed bandwidth storage scheme recommended as a viable means of surmounting this difficulty. It is suggested that advantage may be gained by constructing the transition rate matrix row by row and performing the reduction step on each row as soon as it is generated. An example which has received considerable attention using numerical iterative methods is analyzed using the proposed direct method and the latter method is shown to be superior.

INTRODUCTION

The purpose of this paper is to demonstrate that contrary to the popularly held belief, direct numerical methods for the solution of queueing networks can sometimes be much superior to the more usually employed iterative methods. We will present a direct numerical method, show how implementation problems may be surmounted and finally make some comparisons on a model which has been subject to considerable analysis by iterative techniques.

From the Chapmann-Kolmogoroff equations we may easily determine the following matrix relation, (see for example, [1]):

$$\underset{\sim}{S}^T \underset{\sim}{P} = 0 \qquad \text{(1)}$$

in which $\underset{\sim}{S}$ is an (n x n) transition rate matrix whose elements

$$s_{ij} = \text{the rate of transition from state } i \text{ to state } j \text{ if } i \neq j.$$

$$= -\sum_{k=i}^{n} s_{ik} \quad \text{if } i = j,$$

$\underset{\sim}{P}$ is the stationary probability vector whose component p_i denotes the long run probability of the system being in state i and n is the total number of states which the markovian process representing the system being modelled, may occupy.

It is generally assumed that the matrix $\underset{\sim}{S}$ is given or may be derived, and the object is to obtain the stationary probability vector $\underset{\sim}{P}$ by solving the system of homogeneous linear equations (1). Alternatively, the problem may be posed as an eigenvalue problem by writing (1) in the form:

$$\underset{\sim}{W}^T\underset{\sim}{P} = \underset{\sim}{P} \qquad \text{(2)}$$

where $\underset{\sim}{W}^T = (\underset{\sim}{S}^T\Delta t + \underset{\sim}{I})$ and Δt is arbitrary.

If Δt is chosen such that $\Delta t \leq (\max_i |s_{ii}|)^{-1}$, then the matrix $\underset{\sim}{W}$ is a stochastic matrix and may be regarded as the transition probability matrix for the discrete time Markov system in which transitions take place at intervals Δt. From the method of construction of this matrix, it may be shown that there always exists a unit eigenvalue and that no other eigenvalue exceeds this in modulus. The required vector $\underset{\sim}{P}$ is therefore the left eigenvector corresponding to the dominant eigenvalue of the stochastic matrix $\underset{\sim}{W}$.

Since Wallace and Rosenberg first presented their Recursive Queue Analyzer (RQA1, [2]), over a decade ago, numerical techniques for the solution of queueing networks have been exclusively iterative in nature. There are several important reasons for the choice of an iterative approach as opposed to a direct approach. Firstly, an examination of the iterative methods usually employed shows that the only operation in which the matrix $\underset{\sim}{S}^T$ and/or $\underset{\sim}{W}^T$ are involved is a multiplication with one or more vectors. This operation does not alter the form of the matrix and thus compact storage schemes which minimize the amount of memory required to store the matrix, and which in addition are well suited to matrix multiplication, may be conveniently implemented. Since the matrices involved are usually large and very sparse, the savings made by such schemes can be very considerable.

On the other hand, during the reduction phase of direct equation solving methods, the elimination of one non-zero element of the matrix often results in the creation of several non-zero elements in positions which previously contained zero. This is called fill-in and not only does it make the organization of a compact storage scheme more difficult since provision must be made for the deletion and the inclusion of elements, but in addition, the amount of fill-in can often be so extensive that available memory is quickly exhausted. Compact storage schemes for direct methods, and the problem of fill-in are taken up in section 3. A successful direct method must incorporate a means of overcoming these difficulties.

Iterative methods have other advantages in that use may be made of good initial approximations to the solution vector, and this is especially beneficial when a series of related experiments is being conducted. Furthermore an iterative

process may be halted once a certain prespecified tolerance criteria has been satisfied. Finally, since the matrix is never altered, the build up of rounding error is, to all intents and purposes, nonexistent.

For these reasons, iterative methods have traditionally been preferred to direct methods. However, iterative methods have a major disadvantage in that often they require a very long time to converge to the desired solution. More advanced iterative techniques such as simultaneous iteration [3] have helped to alleviate this problem but much research still remains to be done, particularly in estimating a priori, the number of iterations, and hence the time, required for convergence. Direct methods have the advantage that an upper bound on the time required to obtain the solution may be determined before the calculation is initiated. More important, for certain classes of problem, direct methods often result in a much more accurate answer being obtained in less time. Since iterative methods will in general require less memory than direct methods, these latter can only be recommended if they obtain the solution in less time. Finally, when choosing pivots during the reduction phase of a direct method, advantage can be taken of the fact that the diagonal elements are defined to be the largest in any row (recall that $s_{ii} = -\sum_{k\neq i}^{n} s_{ik}$).

2. A Direct Solution Method.

For a non-trivial solution to the set of homogeneous linear equations

$$\underset{\sim}{S}^T \underset{\sim}{p} = 0$$

the matrix $\underset{\sim}{S}$ must be singular and hence ill-conditioned as regards equation solving. However, in general the markovian model will be ergodic (i.e. the associated stochastic matrix will be irreducible and consequently the matrix $\underset{\sim}{S}$ will possess a unique zero eigenvalue), so that it is sufficient to replace one of the rows of $\underset{\sim}{S}^T$ with the vector $\underset{\sim}{1} = (1, 1, \ldots, 1)$ of length n, and to set the corresponding element of the right-hand side also equal to unity. This is equivalent to normalising the solution vector so that the sum of all of its elements equals 1. It is usual to replace the last row of $\underset{\sim}{S}^T$ in this fashion since this will not cause any additional fill-in which would later require to be eliminated, and also because the right-hand side may be ignored until the back substitution is initiated.

An alternative approach, and one which possesses some advantage over that outlined above, is the method of inverse iteration, [4], which we shall now briefly discuss. Consider an iterative scheme based on the relationship

$$\underset{\sim}{p}^{(k)} = (\underset{\sim}{S}^T - \mu \underset{\sim}{I})^{-1} \underset{\sim}{p}^{(k-1)}$$

$\underset{\sim}{p}^{(0)}$ is arbitrary and may be written in the form

$$\underset{\sim}{p}^{(0)} = \sum_{i=1}^{n} \alpha_i \underset{\sim}{q}_i$$

where the $\underset{\sim}{q}_i$ are the right eigenvectors of the matrix $\underset{\sim}{S}^T$ corresponding to eigenvalues λ_i.

Then

$$\begin{aligned} \underset{\sim}{p}^{(k)} &= (\underset{\sim}{S}^T - \mu \underset{\sim}{I})^{-k} \underset{\sim}{p}^{(0)} \\ &= \sum_{i=1}^{n} \alpha_i (\lambda_i - \mu)^{-k} \underset{\sim}{q}_i \\ &= (\lambda_r - \mu)^{-k} \left\{ \alpha_r \underset{\sim}{q}_r + \sum_{i \neq r}^{n} \alpha_i (\lambda_r - \mu)^k (\lambda_i - \mu)^{-k} \underset{\sim}{q}_i \right\} \end{aligned} \qquad (3)$$

Consequently, if for all $i \neq r$, $|\lambda_r - \mu| << |\lambda_i - \mu|$ convergence to the eigenvector $\underset{\sim}{q}_r$ is rapid. If $\mu = \lambda_r$, then the summation in equation (3) equals zero and the vector $\underset{\sim}{q}_r$ will be obtained to full machine precision in a single iteration.

It is usually recommended that instead of forming the inverse of the shifted matrix and then postmultiplying it with the trial vector as indicated in the recurrence formula, inverse iteration be conducted by solving the set of linear equation

$$(\underset{\sim}{S}^T - \mu \underset{\sim}{I}) \underset{\sim}{p}^{(k)} = \underset{\sim}{p}^{(k-1)}$$

If $\mu = \lambda_r$, then it is simply sufficient to replace the zero pivot which arises due to the singularity of the matrix by a small value ε. This should be chosen to be the smallest number for which $1 + \varepsilon > 1$ on the particular computer being used. This results in a very inaccurate solution to the set of equations but a rigorous error analysis, [4], will show that since the elements of the solution vector possess errors in the same ratio, normalizing this vector will yield a very accurate eigenvector.

This approach has an advantage over the first mentioned method in that an estimation of the build-up of rounding error may be obtained. Theoretically, it is known that we should obtain a zero pivot during the reduction of the final row. However, due to rounding error, this will hardly ever be exactly zero; its non-zero value will yield an indication of the rounding error build-up. This is important when very large matrices are being handled on computers with a small word size, for it is known that occasionally the rounding error becomes so large that it swamps the correct solution vector.

Inverse iteration has another advantage in that it requires less arithmetic operations. Replacing the last row of $\underset{\sim}{S}^T$ by $\underset{\sim}{1}$ requires that the first (n-1) elements of this vector be eliminated, each elimination requiring a certain number of

multiplications, additions and a division. The number of elements to be eliminated in the final row using inverse iteration will be substantially less than (n-1). Further, the normalization requirement in inverse iteration requires only (n-1) additions and usually a much smaller number of divisions. We will see in section 4 that the reduction of all (n-1) elements in the first method entails a further disadvantage in that this elimination requires access to be available to the (n-1) previous rows of the matrix.

3. Compact Storage Schemes for Direct Methods.

The size of the matrices generated by problems of the type considered in this paper are often too large to permit regular two-dimensional arrays to be used to store the matrices in computer memory. Since these matrices are usually very sparse, it is economical to use some sort of packing scheme whereby only the non-zero elements and their position in the matrix are stored. However, when a direct equation solving method is to be applied, provision must be made to include elements which become non-zero during the reduction and somewhat less important, to delete elements which have been eliminated. If memory locations are not urgently required, the easiest way of deleting an element is to set it to zero without trying to recuperate the words which were used to store the element and its location pointers. To include an element into the storage scheme, either some means of appending this element to the end of the storage arrays must be provided, or else sufficient space must be left throughout the arrays so that fill-in can be accommodated as and when it occurs. The first usually requires the use of link pointers and is most useful if the non-zero elements are randomly dispersed throughout the matrix, while the second is more useful if the pattern of non-zero elements is rather regular.

An example of a (4 x 4) matrix stored in compact form using address links is given below:

$$A = \begin{bmatrix} -2.1 & 0.0 & 1.7 & 0.4 \\ 0.8 & -0.8 & 0.0 & 0.0 \\ 0.2 & 1.5 & -1.7 & 0.0 \\ 0.0 & 0.3 & 0.2 & -0.5 \end{bmatrix}$$

Real array	A:	-2.1	-0.8	-1.7	-0.5	1.7	0.4	0.8	0.2	1.5	0.3	0.2
Row array	RA:	1	2	3	4	1	1	2	3	3	4	4
Column array	CA:	1	2	3	4	3	4	1	1	2	2	3
Link array	LA:	5	8	10	0	6	7	2	9	3	11	4

The non-zero elements of $\underset{\sim}{A}$ are stored in any order in the real array A and their row and column positions are stored respectively in the integer arrays RA and CA. In this particular example the link has been constructed so that the

non-zero elements can be accessed in a row-wise sense, i.e. the value denoted in the integer link array of any non-zero element points to the position in the real array A at which the next non-zero element in the row may be found. The last non-zero element in any row points to the first in the following row. Normally it is useful if the chain can be entered at several points; this is achieved in this example by listing the diagonal elements first in the array. To see how an element may be included, consider the elimination of the element in position (2,1) which causes 0.442 to be added into position (2,3) which was previously empty. This is handled on this storage scheme by simply appending

A:	0.442
RA:	2
CA:	3
LA:	8

to the arrays. Note that the links must be updated so that the link which previously indicated 8 (i.e. the second element) now points to 12. This updating in fact constitutes a major disadvantage of this type of storage scheme since it is not unusual for it to require more computation time than the actual operations involved in the reduction. A second disadvantage is the fact that three integer arrays are required in addition to the array which contains the non-zero elements.

Regular pattern storage schemes do not suffer from these drawbacks and may be used if the non-zero elements of the matrix occur in a well defined manner; the pattern of non-zero elements dictates the particular storage scheme to be used. In queueing networks, the non-zero elements often lie relatively close to the diagonal so that a fixed bandwidth scheme may be used. As an example, we show below how a (6 x 6) matrix may be stored using a fixed bandwidth scheme of size 3.

$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 & 0 & 0 \\ 0 & 0 & a_{33} & 0 & 0 & 0 \\ 0 & 0 & a_{43} & a_{44} & a_{45} & 0 \\ 0 & 0 & 0 & 0 & a_{55} & a_{56} \\ 0 & 0 & 0 & 0 & a_{65} & a_{66} \end{bmatrix} \Longrightarrow \begin{bmatrix} 0 & a_{11} & a_{12} \\ a_{21} & a_{22} & 0 \\ 0 & a_{33} & 0 \\ a_{43} & a_{44} & a_{45} \\ 0 & a_{55} & a_{56} \\ a_{65} & a_{66} & 0 \end{bmatrix}$$

Matrix operations in general and equation solving in particular, can be programmed with virtually the same ease using these regular pattern storage schemes as they can be using standard matrix storage. It is easy to show that any fill-in which occurs is restricted to this band. Furthermore, there is no storage requirement for secondary arrays and neither is there any computation time used in the processing of such arrays. It is therefore advantageous to adopt these types of storage where possible. Even where some of the elements within the regular pattern

are zero, it may be more economical and convenient to use a regular pattern scheme than a random or systemmatic scheme. Alternatively a somewhat similar technique, called the variable bandwidth method may be employed. Further information on this method may be found in Jennings, [5], the fixed bandwidth scheme being sufficient for our present purposes.

4. Simultaneous Row Generation and Reduction.

When applying direct equation solving methods such as Gaussian elimination, it is usually assumed that the complete set of linear equations has already been derived and that the entire coefficient matrix is stored somewhere in the computer memory, albeit in a compact form. The reduction phase begins by using the first equation to eliminate all non-zero elements in the first column of the coefficient matrix from column position 2 through n. More generally, during the i-th reduction step, the i-th equation is used to eliminate all non-zero elements in the i-th column from positions (i+1) through n. (Naturally, it is assumed that the pivot elements are always non-zero, otherwise the reduction breaks down).

However, since we are responsible for both the initial generation of the system of equations and for its solution, it is possible to envisage an alternative approach, and one which has several advantages over the traditional method outlined above. Assume, as is usually the case, that the coefficient matrix is derived row by row. Then, immediately after the second row has been obtained, it is possible to eliminate its sub-diagonal element in position (2,1) by adding a multiple of the first row to it. This process may be continued recursively so that when the i-th row of the coefficient matrix is generated, rows 1 through (i-1) will already have been derived and reduced to upper triangular form. The first (i-1) rows may therefore be used to eliminate all non-zero elements in row i from column positions (i, 1) through (i,i-1), thus putting it into the desired triangular form.

This method has a distinct advantage in that once a row has been generated in this fashion, no more fill-in will occur into this row. It is suggested that a separate storage area be reserved to hold temporarily a single initial (i.e. unreduced) row, and the reduction may be performed here. Once completed, the reduced row may be compacted into any convenient form and appended to the rows which have already been reduced. In this way no storage space is wasted holding subdiagonal elements which, due to elimination, have become zero, nor in reserving space for the inclusion of additional elements. The storage scheme should be chosen bearing in mind the fact that these rows will be used in the reduction of further rows and also later in the algorithm during the backsubstitution phase. If the non-zero elements of the coefficient matrix lie along lines which run parallel to the diagonal, (this often arises in queueing networks with two stations only), then it is

probable that there will be considerable fill-in from the diagonal element to the furthermost right-hand element and consequently a fixed bandwidth scheme is likely to be most suitable. On the other hand, if it is known that fill-in will not be extensive, a semi-systematic packing scheme such as is used in iterative methods, [6], may be profitably employed.

If the matrix is band shaped and very large, (e.g. a simple queueing network with a large maximum number of customers), then this approach has a further advantage. When available fast memory is exhausted, it is convenient to put a large section of the reduced matrix onto backing store and this will need to be returned to fast memory only once, i.e. for the final back substitution phase. Consider the matrix shown below in figure 1 and assume that available fast memory can only hold ℓ reduced lines. Let h denote the maximum number of non-zero elements to the left of the diagonal element.

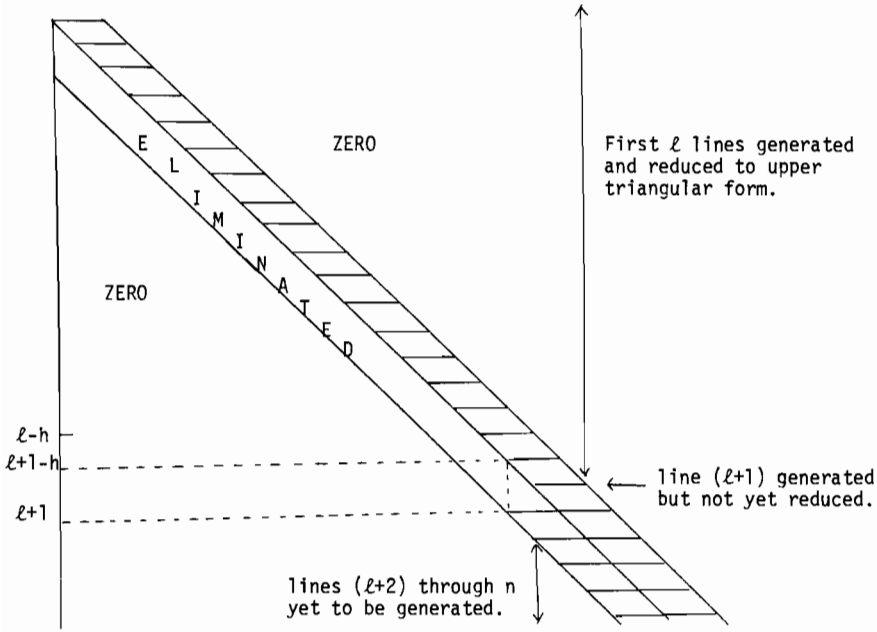


Figure 1. Shows that first (ℓ-h) rows may now be put on backing store.

Then once the ℓ-th row has been generated, reduced and appended to the rows already reduced there will remain no available fast memory locations for the reduced form of lines (ℓ+1) through n. However, since rows ℓ through (ℓ-h) are no longer required for the reduction of rows below the ℓ-th, they may be put onto backing store and the memory thus made available may be used to store a further (ℓ-h) reduced rows. Depending on the size of the matrix and available fast memory, this process may have to be performed a number of times. Note that if the last row of the matrix was defined to be $\underset{\sim}{1}$, its reduction implies that access must be available to all rows from 1 to (n-1) and consequently an additional access will have to be made to rows which have been put on backing store.

Since we require the solution of $\underset{\sim}{S}^T\underset{\sim}{P} = 0$, generating the rows of $\underset{\sim}{S}^T$ is equivalent to generating the columns of $\underset{\sim}{S}$. (Recall that the element $s_{k\ell}$ of $\underset{\sim}{S}$ is defined to be the rate of transition from state k to state ℓ). This is the opposite to the procedure which is normally adopted; viz: from any state k it is usual to determine the states which can be reached in a single time step from this state, and thereby obtain row k of $\underset{\sim}{S}$. To obtain column k of $\underset{\sim}{S}$ it is necessary to inverse this procedure, and to determine from which states can state k be reached in a single time step, and the corresponding rates of transition. However, computationally, the two procedures are identical.

5. Test Results.

As an example of the use of the direct method, we will consider the numerical solution of the queue $\lambda(m)/K/r$, i.e. a queue in which the arrival process is poissonian with arrival rates $\lambda(m)$ which depend on the number of customers in the station; the parameter K indicates a general service time distribution which has a rational Laplace transform, and the station itself contains r identical servers. It is assumed that there exists an integer M such that $\lambda(m) > 0$ for all $m < M$ and $\lambda(m) = 0$ for all $m \geq M$. This queue has been subject to considerable analysis using numerical iterative methods, (Stewart and Marie, [6]) and it is therefore useful to compare their results against those obtained in this paper.

It is shown in the reference quoted above that the transition rate matrix for this queue, when each of the r servers is represented by a law of Cox, [7], has the block structure presented in figure 2, the dimension of each block other than the initial block being $\binom{r+k-1}{r} \times \binom{r+k-1}{r}$. The total number of states generated, n (which is equal to the size of the matrix) is given by

$$n = \sum_{i=0}^{r} \binom{i+k-1}{i} + (M - r) \binom{r+k-1}{r}$$

This block structure is well suited to a fixed or variable bandwidth storage scheme, the maximum bandwidth for an unreduced row being $3 \times \binom{r+k-1}{r}$ and $2 \times \binom{r+k-1}{r}$

once the row has been reduced. Under special circumstances, for example if the servers are all Erlang, then it may be shown that the maximum bandwidth for reduced rows may be decreased even further. A variable bandwidth would permit a saving of $(M - r)\left\{\binom{r+k-1}{r}\right\}^2/2$ memory locations over the standard fixed bandwidth but is a little more complicated to organize. In the examples presented below, a fixed bandwidth storage scheme was employed and since only Erlang servers were modelled, the bandwidth for reduced rows was taken to be $\binom{r+k}{r+1}$ which is less than $2 \times \binom{r+k-1}{r}$.

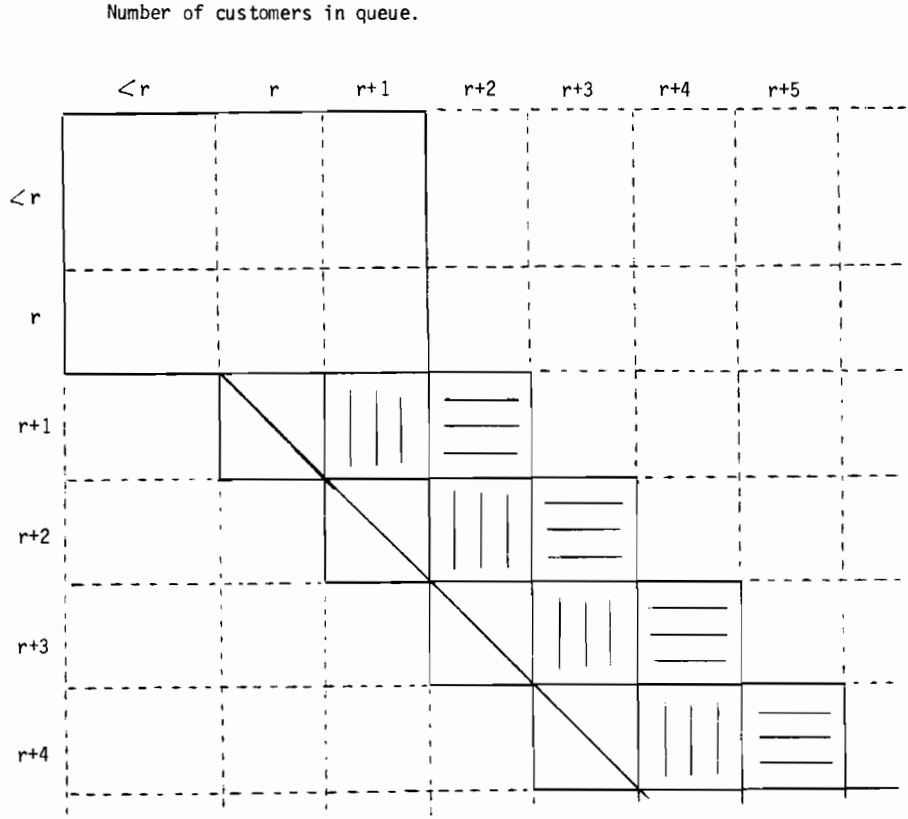


Figure 2. Block Tridiagonal form of transition rate matrix S^T.

The approximate total memory requirements for the direct approach is therefore given by $\binom{r+k}{r+1}$ x n words for storing the matrix and 1.5K words for the actual programming. The programming requirements for the iterative method are much larger, (17.5K words) since the iterative program is much more complicated than the method. However, the number of memory locations required to handle the arrays is approximately equal to 11n, if as is usually the case, five trial vectors are employed in the simultaneous iteration method. In general then, the iterative approach requires less memory than the direct. Note that it is assumed in both cases that backing store is not used.

Figures 3 and 4 show the time required by both methods as the number of identical servers r in the queue is increased. The servers are taken to be Erlang-3 and Erlang-4 respectively.

Figures 5 and 6 show the time required by the two methods when the number of servers is constant, (equal to two and three respectively,) and the parameter k of the Erlang distribution law Erlang E_k is varied.

It will be observed that in all cases the direct approach requires considerably less computation time than the iterative method. Furthermore its results were obtained to full machine precision. Although no timing experiments were conducted to compare the effect of increasing the maximum number of customers M in the queue (this was taken to be ten in all the examples), it is very likely that the direct method will prove to be superior to the iterative method. A moment's reflection will show that as M increases the computation time of the direct method will increase only linearly. No such observation can be made for iterative methods and in fact experiment has shown that often this is not the case.

For the iterative method, initial approximations to the solution vector were obtained by using an analytic technique to determine the probability distribution of customers $\bar{P}(m)$, in the queue when the service distribution is exponential rather than Erlang-k. An approximation to the stationary probability of any state which has m customers is then given by $P(m)/\delta$, where δ denotes the total number of states having m customers.

As an aside we should note that the iterative solution to this queue was developed for inclusion into an approximate method for the solution of networks of queues in which the stations contain more than one general server, Marie and Stewart, [8]. This approximate method embeds the numerical iterative technique into a global iterative procedure, so that the results obtained from one global iteration may be supplied as initial approximations for the numerical method during the next global iteration. After one or two such iterations, it is found that the initial approximations to the solution are very good and therefore convergence is very rapid.

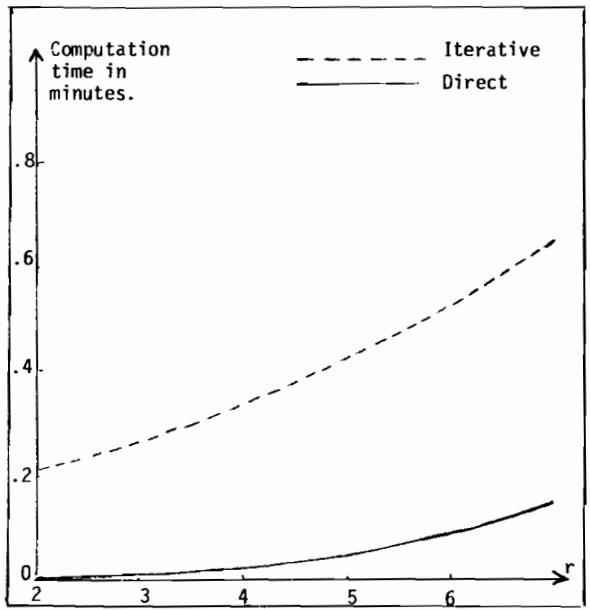


Figure 3. Erlang-3.

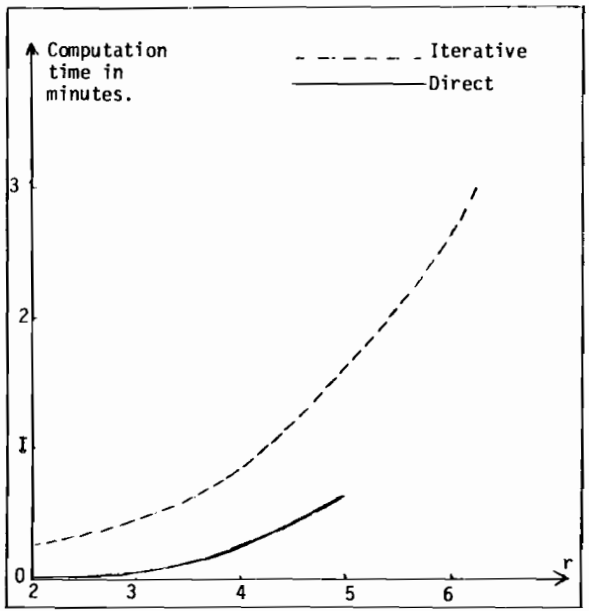


Figure 4. Erlang-4.

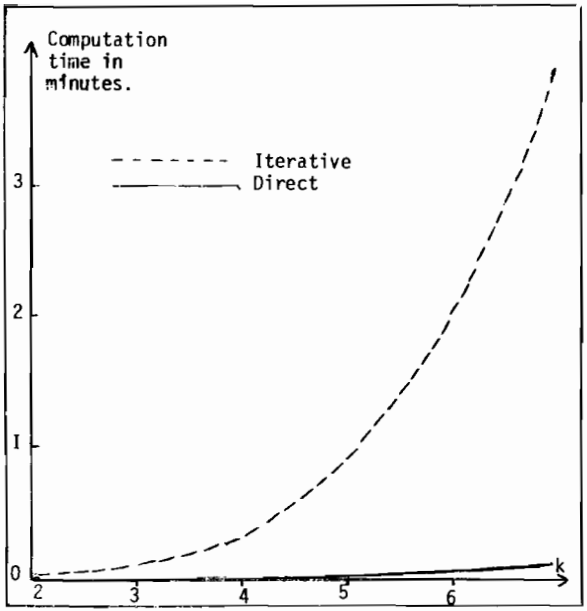


Figure 5. Queue with 2 identical servers.

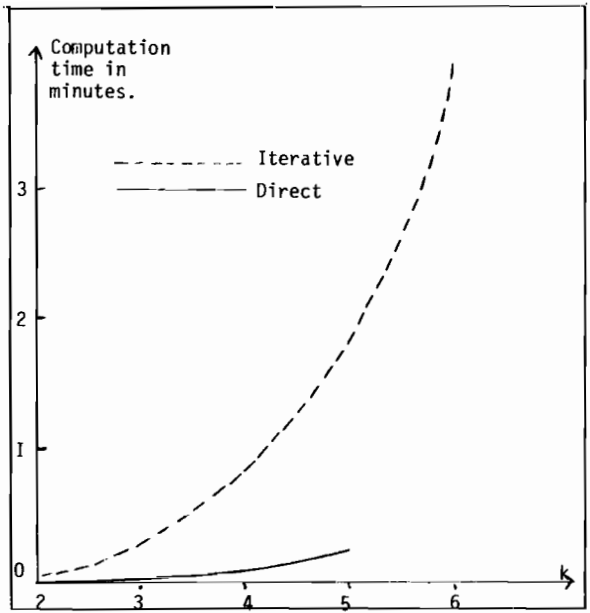


Figure 6. Queue with 3 identical servers.

6. Conclusions.

In this paper we presented a direct numerical technique for the solution of queueing networks. We saw that the method of inverse iteration possessed some advantages over the more usual direct methods in that it requires less numerical computation and that it yields a measure of the rounding error involved in the reduction phase of the algorithm. Some sparse storage techniques suitable for a direct equation solving method were then examined and a fixed bandwidth scheme recommended. It was also shown how an efficient method for generating a row of the matrix and reducing it to upper trianglular form before generating the following row could be implemented with advantage. Finally, the results obtained from this method when applied to the $\lambda(m)/K/r$ queue were compared with those obtained from an iterative method. It was observed that although the direct method required more memory for storing arrays, it obtained much more accurate results in a considerably shorter time period.

REFERENCES

1. W.J. STEWART. "A Comparison of Numerical Techniques in Markov Modelling". Comm ACM. Vol. 21, No. 2, pp 144 - 152, Feb. 1978.

2. V. L. WALLACE and R. S. ROSENBERG. "The Recursive Queue Analyzer". System Engineering Dept. Technical Report No. 2, University of Michigan, Ann Arbor, 1966.

3. A. JENNINGS and W. J. STEWART. "Simultaneous Iteration for Partial Eigensolution of Real Matrices". J.I.M.A. Vol. 15, pp 351 - 361. 1975.

4. J. H. WILKINSON. "The Algebraic Eigenvalue Problem". Clarendon Press, Oxford, 1965.

5. A. JENNINGS. "Matrix Computation for Engineers and Scientists". Wiley Interscience Publication, London, 1977.

6. W. J. STEWART and R. MARIE. "A Numerical Solution for the $\lambda(n)/K/r$ Queue". IRISA Publication Interne No. 79, Université et INSA de Rennes, 35031, France, 1977.

7. D. R. COX. "A Use of Complex Probabilities in the Theory of Stochastic Processes". Proc. Camb. Phil. Soc., Vol. 51, pp 313 - 319, 1955.

8. R. MARIE and W.J. STEWART. "A Hybrid Iterative-Numerical Method for the solution of a General Queueing Network". Third International Symposium on Modeling and Performance Evaluation of Computer Systems. Bonn, W.Germany, October 1977.

APPLIED PERFORMANCE ANALYSIS

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

Performance Evaluation of the BASIS System

by

R.P. van de Riet
vakgroep informatica
Vrije Universiteit, Amsterdam

BASIS is an interactive system, based on PASCAL, for the workshop of the introductory course in informatics. It has built-in facilities for evaluating itself and the performance of the students. The aim of the performance evaluation is to have a tool by means of which the system and the course can be gradually improved.

1. INTRODUCTION

The BASIS system is used in the introductory course in informatics as the primary tool for the student's practical work. It is an interactive system for both program composition and program testing. The language is a subset of PASCAL [1] (no records, no sets, no subranges, no pointers, only one data file and no goto's). The only way a student can make a program is by making procedures which can be individually tested. In fact, for the student a program is just the collection of variables and procedures he introduced. The emphasis is on structured programming with short, well documented, procedures. The current BASIS version checks if the procedure text conforms to a simple but adequate lay-out structure and also whether the text contains any form of comment. The editor is a large subset of the UNIX editor [10].

In two preceding IFIP conferences we reported about the design criteria and plans [5] and about the implementation of the system [6]. In this paper we want to discuss several measurements which have been carried out. These measurements concern primarily the functioning of the system in response to the student and vice versa. A major objective which we want to realize with the system is that of more or less automatic upgrading. Not in the sense that bugs are removed (actually the current version of the system is very stable), but in the sense that reactions of the system to student behaviour are improved.

There are several ways to measure the system-student responses in order to make improvement possible. One method is to question the students about the system by means of questionnaires. In an early stage of the development of the system this has been carried out by two psychology students on a group of alpha students (from the humanities). Very few problems were signalled in this way which were not already known by personal communication. In particular, the placement of the semicolon and the use of the editor turned out to be troublesome. This way of measuring the system was not pursued any longer; although it is not impossible to redo such an investigation in the future if some psychologists show interest.
Another method, which will be extensively reported in this paper, is to automatically analyze the conversation between system and student. In this way it turns out to be possible to get a clear picture of what an average student does, how he reacts upon errors, which errors he makes, which constructs he uses, etc. and of the behaviour of the system in terms of response time, error messages (whether they are clumsy or not), etc.

The structure of this paper is as follows. In section 2, we will demonstrate a typical session of a student. In section 3, we will show how a so-called statfile is constructed from the system-student conversation. In section 4, we analyze the global behaviour of the student and we compare several types of students: informatics and mathematics, biology and geology, and students from the humanities. In section 5, a detailed analysis of the errors will be given in terms of reaction time, think time, frequency, adequacy of help information and repetition of errors. Several correlation coefficients will be given also for the different groups mentioned above. In section 6, we will outline how the errors are distributed in time. In section 7, we will report about the analysis which has been performed for each individual error and how this influenced a new version of the system. In section 8, the use of the language constructs will be described. Tnis analysis reflects a little bit the analysis of Knuth [4] about use of FORTRAN constructs and the analysis of Tanenbaum [9] concerning the use of PASCAL constructs. These investigations were designed for optimization purposes of compilers and underlying machines.

Our goal is to know what the student is doing from a pedagogical standpoint so that the course can be improved pedagogically. In this sense our investigations are in the same area as the studies of Sime and Guest [8] where they measure the use of certain language constructs as e.g. if-then-else versus goto's, or the investigations of Gannon [2] who describes several controlled experiments where programmers use (more or less) structured programming tools. In section 9, we report the results of the measurements concerning the system responses, together with a short overview of past measurements by M. Kersten [3] concerning the internal functioning of the system components. Here, we also give some numbers concerning size and speed of the system and the hardware configuration. Finally, in section 10 we describe some future plans.

2. AN EXAMPLE OF A BASIS SESSION

We suppose that the student is somewhere in the middle of the course so that he is already familiar with the notion of variables, types, values, procedures, editing, etc. He will work on a problem where the main procedures have been thought out and written at home. (In fact the course assistants take care that the students do their homework at home and not behind the terminal).
The problem is to calculate the n-th Fibonacci number f[n], defined as f[0] = 0, f[1] = 1, f[i] = f[i-1] + f[i-2], for i > 1.
This problem is identified as exercise1. A possible interaction is shown below. BASIS normally ends its response with an arrow " -->" after which the student gives a next command. If the student types in a procedure (or function), then the BASIS reaction upon a new line is "...", so the student can easily see if he is still typing in the procedure or that he has finished the procedure. In general, BASIS responds with "..." if the command is not finished.

When the student is editing a procedure, with the name "proc", by means of the command "edit(proc)", BASIS responds with "..>", if a new edit command is expected; if the edit command is not ready, as in the case of a(ppend). BASIS responds with "...".

If BASIS detects an error (syntactical or run-time) it responds with showing the line last treated (which can be the command typed in or a line of a procedure) underlining the symbol last treated. Only if the student types in "help" will BASIS respond with "***" followed by an error message. It is possible that the student asks for more help by typing in "help" again. In that case BASIS responds with some global information about the error such as a reference to the manual or some examples. Note that by explicitly asking the student for "help" instead of automatically providing him or her with the error message, we have given the system a tool to measure the student's reaction upon errors. It is now possible to measure the effect of an error message on the number of times the same error is

repeated immediately, or on the time a student needs before he repairs the error (i.e. think time). Furthermore, it saves time for the student (and paper of the terminal) since in quite a number of cases (in fact 70-80 %) he apparently did not ask for the error information.

In BASIS, comment is placed between "(*" and "*)" and may be placed in commands or in the text of a procedure. We will use it to explain what the student is doing. Let us return now to the problem mentioned above. The student starts as follows after logging in under UNIX [10].

```
*** Basis-version 230178, see: 'Basis-handleiding dec 1977'
--> var f0,f1:integer;
... type far=array[1..n]of integer;
    type far=array[1..n]of integer;
                        =
--> help
*** arraybound must be an integral number
--> type far=array[1..10]of integer;
... const n=10;
... f:far;
    f:far;
     =
--> help
*** "=" expected
--> help
*** ex. const month='augustus'
*** ex. const min=10; max=20;
***           maxint=32767
*** see manual 4.1 ( const-declaration )
--> var f:far
... f0:=0;f1:=1;f[2]:=f0+f1;f[3]:=f[2]+f1;write(f[3]);
          2
--> f[4]:=f[3]+f[2];f[5]:=f[4]+f[3];write(f[5]);
          5
--> procedure fibon(n:integer);
... var i:integer;
... begin
...     while i <= n do
...     begin f[i]:=f[i-1]+f[i-2] end;
... write(f[n])
... end
--> fibon(5);
*** remember: layout not correct, see: additions to the manual
*** remember: pr- or fu-declaration without comment

    *** in: fibon
    4       while i<=n do
                    ==
    n           = 5
    i           = <undef>

--> edit(fibon)
..> 1
    procedure fibon(n:integer);
..> a
... (*computes and prints n-th fibonacci number*)
... .
..> /begin f/
        begin f[i]:= f[i-1]+f[i-2] end;
..> s/end;/;i:=i+1/p
```

```
        begin f[i]:= f[i-1]+f[i-2]; i:= i+1
..> a
...     end;
... .
..> /write/
    write(f[n])
..> s/w/    w/p
        write(f[n])
..> 1,$p
    procedure fibon(n:integer);
    (*computes and prints n-th fibonacci number*)
    var i:integer;
    begin
        while i<=n do
        begin f[i]:= f[i-1]+f[i-2]; i:= i+1
        end;
        write(f[n])
   .end
..> q
--> (*We are out of the editor now*)
--> fibon(5);
    *** in: fibon
    5       while i<=n do
                  ==
    n           = 5
    i           = <undef>

--> help
*** variable did not get a value
--> edit(fibon)
..> /begin/
    begin
..> s/n/n i:=2; f[0]:= f0;f[1]:=f1;/p
    begin i:= 2; f[0]:= f0; f[1]:= f1;
..> q
--> fibon(5)
    *** in: fibon
    4  begin i:= 2; f[0]:= f0; f[1]:= f1;
                       =
    n           = 5
    i           = 2

--> list(type)
    type far = array [1..10] of integer;

--> type far = array[0..10] of integer;
    type far = array[0..10] of integer;
         ===
--> help
*** new identifier expected
--> erase(far)
--> type far = array[0..10]of integer;
... var f:far;
... fibon(5)
          5
--> fibon(7)
         13
--> fibon(10)
        55
--> fibon(15)
```

```
    *** in: fibon
    6       begin f[i]:= f[i-1]+f[i-2]; i:= i+1
                    =
    n           = 15
    i           = 11

--> help
*** index of array bound too large
```

Here we skip over the session until the student has made a procedure which produces the fibonacci numbers as real numbers instead of integers.

```
... fibon(10)
          55.0
--> fibon(20)
        6765.0
--> fibon(40)
      1.0233e8
--> fibon(100)
     3.5423e20
--> (*this computation needed 130 real seconds*)
--> save;
--> stop
    cptime: 28.42 sec.
*** end of basissession
```

3. THE STAT-FILE

From the conversation, as shown in the preceding section, a file is constructed provided with some extra information consisting of real time and cpu-time used, which is called stat-file and which is used for gathering the statistics. Obviously, it is not necessary to put on the stat-file most of the system responses as "-->", full error messages and results of computations. Instead, only the error numbers are shown. Furthermore, in order to simplify the analysis of the stat-file the beginning of a procedure declaration is signalled by a "p" and the beginning of an edit session by an "e". The two numbers with which most of the lines start are real time, measured in seconds and cpu-time measured in 20 milliseconds. These numbers are produced at the moment that the line on which they occur is sent from the BASIS system to UNIX to be put on the file. For an input line this is the moment that all the characters are put in a buffer just prior to processing the line. For an output line it is the moment that all processing is done and the line is shown on the terminal.
The stat-file of the preceding section has the following form.

```
03966 26029 login
00032 00046 var f0,f1:integer;
00072 00049 type far=array[1..n]of integer;
#2103
00072 00050
00076 00051 help
00103 00054 type far=array[1..10]of integer;
00123 00059 const n=10;
00214 00060 f:far;
#2003
00214 00060
00218 00061 help
00225 00066 help
00276 00070 var f:far
00347 00071 f0:=0;f1:=1;f[2]:=f0+f1;f[3]:=f[2]+f1;write(f[3]);
```

```
00398 00078 f[4]:=f[3]+f[2];f[5]:=f[4]+f[3];write(f[5]);
00418 00086 procedure fibon(n:integer);
p
00427 00090 var i:integer;
00432 00092 begin
00445 00093     while i <= n do
00490 00095     begin f[i]:=f[i-1]+f[i-2] end;
00499 00099 write(f[n])
00503 00101 end
!
00518 00101 fibon(5);
#4204
00518 00111
00554 00114 edit(fibon)
e
00558 00116 1
00563 00117 a
00595 00117 (*computes and prints n-th fibonacci number*)
00601 00121 .
00612 00121 /begin f/
00630 00130 s/end;/; i:= i+1/p
00640 00144 a
00652 00144     end;
00653 00145 .
00660 00145 /write/
00695 00149 s/w/    w/p
00707 00154 1,$p
00723 00164 q
!
00750 00165 (*We are out of the editor now*)
00764 00168 fibon(5);
#4204
00764 00174
00777 00179 help
00793 00190 edit(fibon)
e
00799 00193 /begin/
00818 00200 s/n/n i:=2; f[0]:= f0;f[1]:=f1;/p
00824 00204 q
!
01098 00298 fibon(5)
#4002
01098 00305
01127 00310 list(type)
01159 00313 type far = array[0..10] of integer;
#2107
01159 00314
01163 00314 help
01190 00320 erase(far)
01209 00321 type far = array[0..10]of integer;
01216 00323 var f:far;
01327 00364 fibon(5)
01339 00385 fibon(7)
01357 00408 fibon(10)
01365 00440 fibon(15)
#4003
01371 00466
01379 00472 help
02766 00964 fibon(10)
02775 00987 fibon(20)
```

```
02783 01038 fibon(40)
02803 01143 fibon(100)
02971 01400 (*this computation needed 130 real seconds*)
02980 01403 save;
02986 01418 stop
=
```

Next follows a list of the keywords and their frequencies as used by the student in procedures. Note that the reproduction of the session here is not complete.

```
02986 01419 integer2
02986 01419 end2
02986 01420 begin2
02986 01420 var1
02986 01420 while1
02986 01421 do1
!
```

4. GLOBAL BEHAVIOUR OF THE STUDENTS

For three courses, one for biology students (biol), one for mathematics students (math) (which includes informatics students) and one for humanities students (human), with 22, 51 and 12 participants, respectively, we analyzed the stat-files from which the following global conclusions can be drawn as depicted in table 1. The table gives numbers for the average student of the three different groups.

One can make the following conclusions from table 1:

1. Mathematics students seem to work most intensively.
2. There is almost no time spent for problem solving; the time is spent for program development, as it should be.
3. Asking one time for "help" seems to help really as the number of repeated errors is smaller with than without "help".
4. This cannot be said for "help-help".
5. A mathematics student spends about 0.023 cpu hours per real hour, which means that, as far as the cpu is concerned, 30 students can simultaneously be connected without problems.
6. The number of errors made per hour seems to be constant: 15.
7. The number of errors made per command seems to be constant: 0.3.

Comparing the humanities students with the biology and mathematics students is not fair for the simple reason that the contents, the exercises and the size of the humanities course differed considerably from the biology and mathematics course. One can get an idea of the amount of work done in the workshop by considering that the biology students had to do 3 a-, 4 b- and 2 c- exercises while the mathematics students did 4 b- and 4 to 5 c-exercises. An a-exercise is very simple such as a procedure which prints the truth table of the operators "and" and "or". A b-exercise is more difficult, for example, a procedure computing a frequency table of letters occurring on an input file. A c-exercise is rather complicated as e.g. tic-tac-toe, simulation of Conway's game of life. The exercises for the humanities students were specially chosen from linguistics, such as text manipulation, e.g., counting letters, justifying text, coding and decoding text and generating text by means of syntactic rules.

	Biol	Math	Human
- Total time connected with BASIS (in hours)	31	22.5	12
- Time spent while editing (in %)	24	28	13
- Time spent while typing procedures (in %)	11	17	13
- Time spent while typing commands (in %) (this includes procedure calls)	33	30	29
- Time spent while thinking about an error (in %)	23	21	3
- Time spent for chatting, coffee drinking etc. (in %)	9	4	42
- Usage of cpu (in hours)	0.74	0.58	0.06
- Number of commands including procedure calls	1404	994	476
- Number of edit calls	186	165	46
- Average number of lines per edit call	9	8	7
- Number of procedures declared	64	36	40
- Average number of lines per procedure	7	12	6
- Number of errors made	447	290	178
- Average time needed before reacting after an error was made (in sec)	59	58	66
- Number of times the same error was immediately made again without asking for help (in %)	28	25	25
- Number of times "help" was called after an error was made (in %)	23	20	29
- Average time that elapsed after an error was made and before "help" was called (in sec)	36	14	28
- Number of times the same error was immediately made again after asking for "help"	18	12	18
- Number of times "help" was called two times after an error was made (in %)	3	6	5
- Average time that elapsed after an error was made and before "help" was called for the second time (in sec)	107	153	124
- Number of times the same error was immediately repeated after asking two times for "help" (in %)	16	16	19

table 1. Global behaviour of students

5. ANALYSIS OF THE ERRORS

For each error, the attributes, as listed in table 2a, were measured for the three courses mentioned:

- n: frequency;
- h: the number of times "help" was called;
- w: the total time elapsed after the error occurred and until a "help" was called; it is called "1- help" wait time;
- hh: the number of times "help" was called twice in succession;
- ww: the total time elapsed after the error occurred and until the second "help" was called; it is called "2- help" wait time;
- e: the number of times the same error was immediately made again alled; after one "help" was called;
- ee: the number of times the same error was immediately made again after two "helps"s were called;
- r: the number of times the same error was immediately made again while neither "help" nor "help help" were called;
- t: the total time elapsed after the error occurred and until any reaction other than "help" or "list" was typed in; this time is called the think time.

table 2a. The total attributes.

From the above attributes concerning total/absolute quantities, we derived the following attributes describing relative quantities:

- mt: mean think time = t/n;
- mh: mean number of "help"s = h/n;
- mhh: mean number of "help - help" per one "help" = hh/h;
- me: mean number of repeated errors after "help" = e/h;
- mee: mean number of repeated errors after "help - help" = ee/hh;
- mr: mean number of repeated errors after neither "help" nor "help - help"; mr = r/(n-h-hh);
- mw: mean "1-help" wait time = w/h;
- mww: mean "2-help" wait time = ww/hh;

table 2b. The relative attributes.

The errors were sorted according to the above attributes, so that we obtained 17 lists of error numbers. It takes too much space to reproduce these lists here since eacn list contains 170 error numbers. We first give a few interesting examples and tnen we will describe the results after comparing these lists.

5.1. Some examples

In order of frequency, n, the 10 most frequently occurring errors were almost the same for the three courses: biology, mathematics and humanities; we therefore list the results of the biology course:

identifier not declared	(typing error)
syntax error	
erroneous symbol	(typing error)
command expected	(typing error)
error in editor witn text replacement	
existing identifier expected	(typing error)
(as e.g. in list or edit)	
variable did not get a value	
editor reaches end of text too early	
after editcommand no new line	
error in string matching in editor	

Tnese 10 errors were responsible for 56% of all errors; four of them probably are

typing errors, four are errors in the editor (which means that working with the editor is still not simple although we changed from an own-invented (clumsy) editor to the very handy UNIX editor) and two are more fundamental errors: one concerning syntax and one concerning semantics.

Errors which need the most time to think before a repairing action are the most important ones. They need considerable attention from the person who gives the lectures as well as from the makers of the system.
In the ordering of total think time, t, the first 10 errors for the biology students were:

variable did not get a value
identifier not declared
syntax error
command expected
identifier not declared
"," or ")" expected after expression
too much cpu-time used
existing identifier expected
erroneous symbol
insufficient room left for array declaration

The mathematics students showed a somewhat similar behaviour; only three of the errors differed. The humanities students showed a more different behaviour, with four errors differing.

The reason that "identifier not declared" is showing up in the above lists is that it occurs so frequently, but it is, of course, a very common error. The most interesting errors are those which need the most mean think time, mt. The attribute mean think time is of interest as it says something about "difficulty".
The ten most "difficult" errors, then, for the biologists were:

insufficient room for array declaration
too much cpu-time used
division by zero
array identifier expected
"(" or identifier expected (in a command of the form "a:= ...", where a is an array variable and "..." can be something like "(1,2,3)" or "b")
type identifier expected
(in "var a: ...")
index of one-dimensional array too small
erroneous use of standard procedure identifier
(as in "procedure sin(x,y:real)"
overflow of real capacity
first index of two-dimensional array too small

The frequencies of these errors range from 3 to 57, which on a total of about 10000 errors is of course very small. The think times range from 6 to 2.6 minutes.

According to the mathematics students the following ten errors were most "difficult":

too much cpu-time used
index of one-dimensional array too small
insufficient room for local array declaration
too much nested procedure calls (max = 50)
(problems with recursive procedures)
array identifier of same type expected
array-bound must be an integer
("type ar = array [1..n] of real" is not allowed)
second index of array is too large

array index must be an integer expression
type of function must be standard
not implemented (probably an error of the kind: "a[1]:= b[1]" occurring in a procedure where a and b are two-dimensional array variables. This construction is allowed as a command, but as it is non-PASCAL, we have forbidden it in a procedure)

It is remarkable that 7 of these errors concern arrays. They were not made often; their frequencies range from 134 to 9 (which is not much compared with the total number of about 15000 errors) and their mean think times range from 10 to 2.3 minutes. With respect to the humanities students we remark that their ten most "difficult" errors differed completely from the above two lists, which is not surprising.

5.2. Some correlations

It is tempting to compare all the sorted lists of errors; the question is, however, how can they be compared? Heuristically, the most direct way is the way we suggested in the preceding section. Compare the first N items of two sorted lists. If the intersection consists of d elements, then d/N is a measure for correspondence. This number is called the correspondence number c(N) and is obviously dependent on N. If N is chosen to be 1, then c(N) is either 0 or 0.5; if N is chosen equal to the number of different errors (170 in our case), then c(N) = 1. We have computed c(N) for N = 10 and N = 20.
For the mathematics students we found the following correspondence numbers in the form of pairs c(10), c(20): (for obvious reasons we donot compare total attributes and relative attributes with each other).

	n	h	w	hh	ww	e	ee	r
n	1, 1							
h	.3, .4	1, 1						
w	.5, .6	.6, .7	1, 1					
hh	.3, .5	.7, .6	.5, .7	1, 1				
ww	.4, .6	.5, .5	.6, .7	.4, .7	1, 1			
e	.3, .3	.8, .7	.6, .5	.7, .5	.5, .5	1, 1		
ee	.4, .3	.6, .6	.4, .4	.5, .6	.5, .5	.5, .6	1, 1	
r	.8, .9	.3, .4	.6, .6	.3, .5	.4, .6	.3, .4	.4, .4	1, 1
t	.6, .7	.7, .7	.7, .8	.6, .6	.5, .7	.7, .5	.5, .5	.5, .7

table 3. Correspondence numbers for total attributes.

The conclusion from table 3 might be that there is in general a rather high correspondence between all the total attributes, but in particular between n and r, w and h, hh and h, e and h, t and w. This conclusion is reinforced by a computation of Pearson's correlation coefficients. (not reproduced here).

The correspondence numbers for the relative attributes are given in table 4.

	mt	mh	mw	mww	mhh	me	mee	mr
mt	1, 1							
mh	0, .2	1, 1						
mw	.2, .2	0, 0	1, 1					
mww	.3, .2	0, 0	.5, .6	1, 1				
mhh	.2, .4	.2, .5	0, .1	0, .1	1, 1			
me	.3, .3	.1, .1	.1, .1	.1, .1	0, .2	1, 1		
mee	0, .1	0, .1	.1, .2	.1, .1	0, .2	.1, .1	1, 1	
mr	0, 0	0, 0	.2, .2	.1, .2	.1 .1	.1, .1	.1, .1	1, 1

table 4. Correspondence numbers for relative attributes.

The conclusion from this table is that not so much can be said: there seems to be a low correspondence between all of the relative attributes. The high correspondence among the total attributes is probably due to the fact that frequencies of the first errors of these lists are very high, whereas the frequencies of the first errors of the relative attribute lists are an order of magnitude smaller, so tnat stochastic effects on the order is much stronger. The only correspondences which can be noted are between mww and mw, which is not very surprising, and between mhh and mh.
The faint correspondence between mt on the one hand side and mww, mhh and me is notified.

Comparing two lists of 170 errors by counting how many errors appear to be common to tne first N (N = 10 and N = 20 above) is of course very arbitrary. We could have taken the last N errors of the lists equally well.
Therefore, we also have worked out the following experiment. Compare the two lists by looking whether the i-th element of the first list occurs on one of the places i-d, i-d+1, ... , i+d-1, i+d in the second list. Such an element is called an OK element. A correspondence number can then be defined as the quotient of the number of OK elements over the total number of elements.
For d = 10 we give here a list of 10 pairs of attributes which have the highest correspondence number:

mee-ee (70%)	tt-h (51%)
hh- ww (58%)	tt-w (51%)
tt- n (57%)	me-e (51%)
h - w (56%)	r -e (50%)
mww-ww (56%)	ee-e (49%)

The lowest correspondence numbers are between mee and mt (11%) and between mee and mh (13%). The highest correspondence numbers between relative attributes are for mhh and mww (40%) and mee and me (41%).
It is noteworthy that mt as attribute does not correspond well with any other relative attribute except maybe with mh (30%) and mw (23%). By means of Pearson's correlation coefficients, another correspondence between the relative attributes has been computed and the results are shown in table 5.

	mt	mh	mw	mww	mhh	me	mee	mr
mt	1							
mh	.3	1						
mw	.4	0	1					
mww	.1	-0.2	.3	1				
mhh	-0.1	0	.1	0	1			
me	0	.1	-0.1	-0.1	-0.1	1		
mee	-0.1	.1	-0.2	-0.2	-0.3	.2	1	
mr	-0.1	.1	0	-0.1	0.1	0	0.3	1

table 5. Pearson's correlation coefficients for relative attributes.

The positive correspondence between the mean think time (mt) on the one hand and mean number of "help" calls (mh) and mean "1-help" wait time (mw) on the other hand is interesting, altnough it is not evident that these attributes are really correlated. The observation that mee and mhh seem to be negatively correlated leads to tne interesting conclusion that the more one does "help-help", the less one makes the same error again. That this conclusion, which certainly is a conclusion with which we would be very happy, should not be drawn too hastily, follows from the fact that a similar conclusion can not be drawn with respect to "help" (the correlation between me and mh even seems to be positive). The same correlation coefficient for the biology course, which was held earlier than the mathematics course, had the value +0.2. After this course was held the error messages for "help-help" were changed, this can be the reason of the negative correlation coefficient, discussed here.
The positive correlation coefficient between mr (mean number of repeated errors without "help") and mee (mean number of repeated errors after "help-help") is notified.

6. THE DISTRIBUTION OF THE ERRORS IN TIME

From Kersten's [3] analysis we give a small account of the analysis of how the errors were distributed in time. Table 6 shows the percentages for six very frequently occurring errors during the three weeks of the course.

	week 1	week 2	week 3
identifier not declared	15.5	11.5	10.2
syntax error	7.6	9.2	7.9
command expected	4.7	7.5	4.2
existing identifier expected	4.4	3.8	5.2
variable did not get a value	1.7	4.3	7.6
error in editor	2.9	3.6	5.3

table 6. Time distribution of most frequent errors.

One can see that simple errors as "identifier not declared" are made less and complicated errors as "variable did not get a value" are made more as the course is going on.

7. ANALYSIS OF EACH INDIVIDUAL ERROR

With tne error frequencies of the biology course we have analyzed each error individually with respect to: adequacy of error message and clarity of the location when the error occurred, and this with the relative importance of the error in mind. For about 20 errors this resulted in a better message and for tnree errors this resulted in reprogramming the pertinent piece of the system. Noteworthy is tne error: "type of operands unequal" which occurs for example in "if i<10 and 1<i", where "10 and 1" is wrongly treated as a term; the reason for this was that we had a large table for all the operators and a three- dimensional "jump" such tnat knowing the types of the operands and the operator immediately led to the table entry where the definition of the operation was defined. We now have dropped this very clever scheme in favour of working out all cases with if-then-else and case-statements with the effect that errors can be better localized; as a side-effect it turned out that the new scheme was faster and needed fewer bytes.

It nas been investigated also why about 30 errors did not show up in the statistics, altnough the system could produce them (by the way, there were 14 errors which occurred just once of the total number of 14800). The conclusion was that in principle they all can (and hence will) occur some time. Some constructs which are possible in BASIS are used very seldom, such as giving through procedure identifiers or call-by- reference parameters to other procedures. There is also a double syntax check: one is rather crude and concerns bracket structure, the other is precise. For example, the first check does not signal an error in "if 0<x<1 then ...", it sees that "then" has a preceding "if". The second check reads "0<x" as an expression and the next symbol to be treated is "<" so that an error is signalled saying that "then" is expected.

8. FREQUENCIES OF KEYWORDS

In order to observe what the students are doing and which constructs they are using we have counted their use of keywords as "begin", "if", etc., using the stat file. This has been performed, for several exercises, with the hope that per exercise tne behaviour of the students is a little bit uniform. for the whole course it is very difficult to draw any conclusion about the usage of keywords. Martin Kersten has performed such an analysis and could only draw conclusions like:

- "and" and "or" are used much more frequently than that boolean variables are being declared;
- "then" is used about twice as often as "else";
- the use of "for", "repeat" and "while" is proportional to 9:1:1.

This topic will be dealt with more extensively in a future paper.

9. THE INTERNAL FUNCTIONING OF THE SYSTEM

9.1. The internal structure and representation

In order to get an idea of the functioning of the system and thereby on the kind of measurements which have been carried out, we give the following global picture of the system and the internal representation.

The system is completely written in PASCAL.
Tne text of each BASIS procedure (i.e. typed in by the student) is kept in memory as a linear list of text cells. Each text cell contains a syntactic unit, such as an identifier or a keyword in which case a pointer in the text cell points to the character string constituting tne identifier or keyword in the symbol table, or a number, in which case a pointer points to a record in the number table, containing tne character string (for editorial reasons "123.456" is different from "1.23456e2") and the value, or the text cell contains a character like ";" or "=" or an end- of-line symbol, in which case backward and forward pointers simplify the process of stepping line-by-line through a text.

There is also an information store in which the information (type and for variables: value and for procedures: text pointer) about all identifiers is stored. There are pointers from the symbol table to this information store in order to find the appropriate information corresponding to an identifier occurring in the text.
By a careful analysis of the frequencies of the records we were able to redesign the internal representation in such a way that the amount of memory needed for a certain program was diminished by a factor of about 2.5. Which means that a program can now have a size of some eight pages in the available space, which is more than enough for a basic course. For example, a text cell consisted in the previous system of 10 to 12 bytes, in the current system it consists of 4 (normal) to 10 (only for end-of-line cells) bytes. In the previous system we encountered the problem that the PASCAL system maintains 6 free lists of records which may be reused. The new system is provided with one record type: cell with 35 (nested) variants and the BASIS system itself maintains lists of free cells structured on size.

Another redesign concerned the symbol table. In the previous system it was organized as a binary tree upon initialization prefilled with the keywords. It turned out that on the average 6 comparisons were necessary to insert or look after an identifier or key word. The storage structure of the current system is that of a nash table of moderate size of 248 bytes. The average number of comparisons now is 1.1.

The system code amounts to 155 PASCAL procedures totaling 3300 lines, having been thoroughly analyzed and judiciously rewritten based on an analysis using counters and timing statistics of Martin Kersten [3]. The result was that the current system is about a factor 1.5 faster than the old one. We plan to describe the internal functioning of the system and the changes we made in more detail in another paper.

9.2. The hard- and software configuration

The hardware configuration consists of:

- PDP 11/45 with 124 K words
- cache memory (speed improvement about 40%)
- floating-point processor
- 2 RK05 disks (2*2.5 M bytes)
- 1 fixed head disk (0.5 M bytes)
- 2 Ampex disks (2*67 M bytes)
- Lineprinter + paper tape reader/punch
- 2 DEC Tape Units
- 30 Terminals
- 1 fast multiplexor for connection to a remote Cyber 73.

As software, the UNIX operating system [10] is in use; this is a time-sharing system allowing processes to share common files. The PASCAL compiler used for the BASIS system is home-made. It produces code in a very compact form, which is executed by interpretation loosing about a factor eight in speed as compared to assembly code. The code for the BASIS system takes 26 K bytes and there are 18.5 K bytes necessary for library, tables and buffers. The system is itself also interpreting, so executing a procedure in BASIS is a time- consuming process. For example, an empty for statement of 100 repetitions takes 0.17 cpu sec and a while-statement 1.8 cpu sec. From the measurements in the next section we will see that executing procedures is done in only 9% of the total time, so the slowness is certainly not prohibitive. It is rather awkward, however, to see a student waiting for a long time while the system is executing his procedure, in particular at the end of the course. Therefore, a new system is under development, which combines the existing UNIX editor and PASCAL compiler to realize a more rapid BASIS system.

Witn regard to the PASCAL system we remark that a new system is almost ready in which the compiler itself takes 14 K bytes only. This system can be used to produce optimized short code or optimized assembly code. The ratios for speed and storage space of short code versus assembly code are roughly the same: four. The BASIS system compiled to assembly-code needs about 50 K bytes and runs two to three times faster than the current system. This BASIS system will be used during the next course so that actual measurements can detect improvements.

9.3. The response time of the system

Using the stat files it is possible to measure some interesting response times. In a typical conversation there are certain times which are of interest. They are shown in fig. 1.

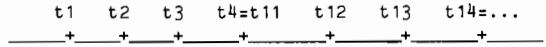


fig. 1 Events during a conversation.

The times are defined as follows:

t1, t11: student starts typing a line of text;

t2, t12: student sends line to the system;

t3, t13: BASIS has seen the line and sends the line, provided with current real time and current cpu-time, to the stat file;

t4=t11, t14: the computation as specified by the line is performed and the reaction of BASIS is sent to the terminal, whereupon the student can start thinking and/or typing a new line thereby repeating the cycle. If the system reports an error this is put, together with the current times, on the stat file.

Evidently, tne time t4-t2 is the response time which we want to measure as a function of the number of other simultaneous users of the system.
Tne following experiments have been carried out.
first, we asked n BASIS users to edit only, typing a line now and then. For n running from 0 to 18 this had no effect on the response time of the (n+1)-st user.
Second, we asked n BASIS users to use the system heavily by executing an infinite loop, again for n=0, ... ,18. The effect on the response time of the (n+1)-st user was measured as follows. The (n+1)-st user was executing a procedure which used about 1 cpu second. The real time was measured this procedure needed to get ready. This was done by computing t13-t3 and neglecting t12-t11 (using the UNIX facility to type one line ahead). The results can reasonably accurately be described by the formula:

$$r = 1.7\ (n+1),$$

where r is the ratio of real time and cpu time.
Tne ideal formula would have been r = n+1; the factor 1.7 is caused by swapping overhead.
The response time for just typing in a new line could not be measured by means of the stat file, since in t3-t2 an unregistered amount of time elapsed before BASIS turns attention to the user. This time, delta, could be measured by simply using a stopwatch. It is given in the following formula:

$$delta = 0.2\ (n+1)\ sec.$$

It is of course also interesting to see from the stat files how long a student really waited in the course for the execution of procedures. Therefore, we computed t4-t2 from t4-t2 = (t13-t3) - (t12-t11) assuming that t13-t12=t2-t2.
The time t12-t11, i.e. think time plus type time, was estimated using cases were the procedure call lead to a syntax error.
The following conclusions could be drawn; they are given in table 8.

	week1	week2	week3	total
total real time waiting for computation (in min)	8	15	129	152
percentage of real time waiting for computation compared to total time connected (in %)	1	5	14	9
total cpu time for procedure computation (in min)	1.7	2.6	15	19.3
percentage of cpu time needed for procedure calls compared to total cpu time used (in %)	50	55	63	62
ratio of real time waiting for computation and the cpu time used for it	4.7	5.8	8.6	7.9

table 8. Response- and waiting times for procedure calls.

The conclusion from table 8 is that during the third week of the course on the average 2.25 students of the 15 simultaneous students were using the system heavily. This would result in a ratio of 3.8 for real time over cpu time a procedure call needs. The actual observed ratio was 8.9. An explanation for the difference is that firstly the students which are editing are using the system more heavily than in the experiment above. Secondly, while the course was going on an unknown number of other people were using UNIX. So the actual ratio of 8.9 is quite reasonable.

10. FUTURE PLANS

As has been said already in section 9.2, a new BASIS system which is built around the existing editor and existing compiler is under development.
Another plan is to direct the measurements on the individual behaviour of the student: is he making too much errors so that he needs help, is he using strange constructs. This is, however, quite complicated since it is then necessary to know the behaviour of a "normal" student. At the moment we only save his last ten errors in order to be able to give him a message when he makes the same error more tnan two times in a certain time period.

ACKNOWLEDGEMENT

The author is very grateful to Ruud Wiggers, for doing all the programming and the measurements and to his student Martin Kersten for numerous ideas and critical remarks. Furthermore, he thanks Anthony I. Wasserman, University of California, San Francisco, and temporarily guest of the vakgroep informatica, for numerous remarks concerning the text of this paper.

LITERATURE

[1] K. Jensen, N. Wirth, PASCAL User Manual and Report, Lecture Notes in Computer Science, Springer, 1974.
[2] John D. Gannon, Language Design to Enhance Program Reliability, Technical Report CSRG-47, January 1975, Computer Systems Research Group, University of Toronto. Also available in SIGPLAN Notices 10,6.
[3] M. Kersten, An Evaluation of the BASIS System, Informatica Report IR-21, Wiskundig Seminarium, Vrije Universiteit, Amsterdam, 1977.
[4] Donald E. Knuth, An Empirical Study of FORTRAN Programs, Software - Practice &

Experience, Vol 1, No. 2 (1977) pp 105-134.
[5] R.P. van de Riet, Some Criteria for Elementary Programming Languages, in Computers in Education, O. Lecarme and R.W. Lewis (Eds), IFIP, North Holland Publishing Company 1975, pp 953-963.
[6] R.P. van de Riet, BASIS - an Interactive System for the Introductory Course in Informatics, Information Processing 77, Proceedings of IFIP Congress 77, North Holland Publishing Company, 1977, pp 347-351.
[7] R.P. van de Riet, R. Wiggers, The Implementation of BASIS, Report IR 13, Wiskundig Seminarium, Vrije Universiteit, Amsterdam, 1976.
[8] M.E. Sime, T.R.G. Green, D.J. Guest, Psychological Evaluation of Two Conditional Constructs used in Computer Languages, Int. J. Man-Machine Studies (1973) 5, pp 105-113.
[9] A.S. Tanenbaum, Implications of Structured Programming for Machine Architecture, Communications of the ACM, Vol 21, No. 3, (March 1978), pp 237-246.
[10] K. Thompson, D.M. Ritchie, The UNIX Time-Sharing System, Communications of the ACM, Vol 17, No. 7, (July 1974), pp 365-375.

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

A MODEL OF A HETEROGENEOUS MULTIPLE-MINICOMPUTER : SYSTEM M2 PERFORMANCE EVALUATION DURING DEVELOPMENTS

J.-J. GUILLEMAUD

Laboratoire de Recherche en Informatique
Université de Paris-Sud
91405 Orsay (France)

and

Ecole Supérieure d'Electricité
Service Informatique
91190 Gif/yvette (France)

The work* reported in this paper is a performance evaluation of M2, a heterogenous multi-mini processor, currently being developped and built at Ecole Supérieure d'Electricité, France. The purpose of this study is to evaluate the main performance bootlenecks of this system, which is meant to replace an existing time-sharing system. M2 is composed of a set of small computers, each operating in mono-programming mode, communicating through a common bus with a master control machine and a file processing machine. Job steps are allocated to a given small machine by the master which carries out allocation decisions and the exchange of messages between the terminals and the job steps. We start from a measurement study of the currently available workload to obtain a projection of the workload that the new system is supposed to support. This information allows us to predict the input traffic to the bus, the frequency of I/O commands to the file processor, and the job step characteristics. A hierarchical model is then developped which allows us to determine the order in which these various ressources saturate, as well as the performance of each individual component and of the system as a whole.

SYSTEM DESCRIPTION

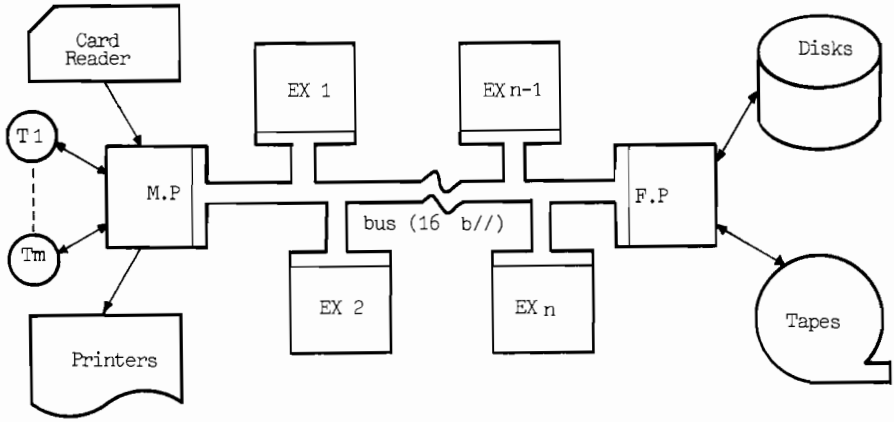
This paper presents a model of an experimental computing system in current development at Ecole Supérieure d'Electricité, Gif/Yvette (France).

This computing system, called M2 for : Multi(ple)Mini(computers) is a closed network of mini-computers, and is designed to be used mostly for interactive processing [7].
Its main features are :

- Transparency : a user at his terminal will see the system M2 as a large and powerful single computer.
- Modularity : from 1 up to 14 independent mini-computers can be enclosed in the network, to attain a desired level of computing power (expected goal : 100 active terminals in an educational environment). As these mini-computers will execute all of the user's tasks, they will be called Executors.
- Heterogeneity : all Executors can be of different types. Thus system M2 can be up graded, step by step, as technology improves, or when the workload justifies it.

The common ressources (see Fig. 1), shared by all Executors are :

* *This work was supported by a scholarship from D.G.R.S.T. (France)*



T 1...Tm : Terminals F.P. : File-Processor

EX 1 EX n : Executors M.P. : Master-Processor

Figure 1 : Multi-Minis (M2)

- a high speed bus, able to transfer data at a 1 M byte/s rate. The transfer unit is a packet whose maximum length is 4K bytes. A transfer cannot be interrupted after it is started (packet switching mode).
- a mini-computer dedicated to file processing (PDP 11/40) - This File-Processor is in charge of all operations on disks and tapes.
- a master-mini-computer which controls all exchanges with the outside world (terminals, printers, etc...). The Master-Processor is responsible for the assignment of job-steps (see-below) to Executors. Usually, this is done in order to equalize the workload between the Executors.

In such a system, the performance of common ressources is of prime importance since it determines the maximum number of Executors which can be connected to the bus.

The current M2 configuration is as follows (see Fig. 1) :

Master-Processor : CII-MITRA 15/35, 64Kb, local disk
File-Processor : DEC-PDP 11/40, 64Kb, driving two disk modules of 88 Mb each
4 Executors : 2 * CII-MITRA 15/35, 64Kb, local disk
2 * SEMS-MITRA 125, 128Kb, local disk

Since Executors may be of different types, there is no common machine language for M2. It would have been possible to emulate a common Executor on each particular Executor but this solution has not been selected because of its very low efficiency. The solution adopted is the following : each job-step is run on one of the Executors, and it will stay on this same Executor till full completion. It is, therefore, necessary to translate the input program code (whatever it is), into the local Executor code, and to do this only once, when a new job-step begins (see Fig. 2). Thus translation overhead will be small enough not to impair system efficiency.

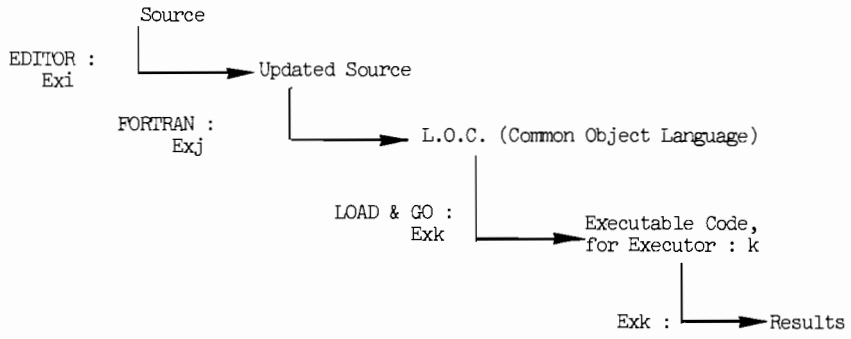


Figure 2 : Code routing, from Source stage to Run stage.

Each Executor is designed so as to support up to 32 active processes. At a given instant, there is only one task in main memory, all ressources of the Executor being then available to this task (mono-programming). The remaining processes assigned to this Executor are stored on a local disk which is used to swap processes in and out of main memory.

PERFORMANCE ANALYSIS OF M2

The following model was developed for performance prediction during design. Later on, it can also be used by the scheduling algorithm to improve the dynamic behaviour of M2 in the presence of time-varying workloads.

We shall first briefly mention the results of a study concerning the bus ; the details will not be given since our results showed that the bus does not constitute a bottleneck or a critical ressource. The main effort of our analysis is turned towards the evaluation of global system performance.

The Bus

To begin with, the maximum workload of M2 was estimated through a detailed analysis of the current measured workload of the existing system (CII 10070, 25 terminals connected) which M2 is supposed to replace. No assumptions were made on the internal structure of M2, except that it should satisfy 4 times more users (i.e. 100 terminals), 4 times faster than the existing system.

Using these measurements, the projected traffic on the bus was estimated, and a maximum utilisation factor of 25% was obtained for the bus. This detailed measurement and modeling study will not be presented here ; it can be found in [5] . This is an important result, since it was then possible to assume a mean response time of the bus comparable to its service or data transfer time.

The File-Processor

File-Processor description

- The current file-processor is a PDP 11/40, connected via a single channel to 2 removable disk-pack modules, much like the IBM 3330, of 88 M bytes each.
- Requests to the File-Processor originate from the Master-Processor and/or the Executors, via the bus. A request can be a high level access function, such as : open-file, read-using-a-key, delete-file, etc.... All requests are serviced at the same priority level. Depending on its type, a request may generate any number of physical accesses to tape or disk. Requests deal only with complete blocs of data ; logical articles are unknown to the File-Processor. Since tape operations will not be allowed to non-privileged users, and should represent a very small part of the load, only disk-files will be considered here. To optimize file accesses, special care is taken to ensure a well performing organization on the disk. Files can be keyed or sequential ; in both cases, blocs inside a file are linked together. When a new file is created, it is assigned to a cylinder ; as long as possible, all sectors of this file will be allocated on this same cylinder. Rotationnal latency is also being minimized : sequentially linked blocs will have successive addresses. The minimal distance between two successive blocs will be such that the File-Processor CPU has enough time to handle intermediate computing. The purpose of this organization is to ensure that, in most cases, a single request will be serviced inside a single cylinder and as fast as possible. Further details are available in [8].

The file-Processor Model

It is shown on Fig. 3. It is modelled as a closed network of queues, which can be solved using Gordon & Newell's method ([4]).

In the model, all service times are represented by exponential distributions, while service policies are supposed to be FIFO. In fact, this is untrue, because arm movements, on each disk-module, are optimized through a Look ([9]) policy, while the channel is allocated on a SLTF basis (Shortest Latency Time First). But as long as we are only interested in throughput, a FIFO approximation will lead to the same results as LOOK, provided that service rates are computed using the true policy. When using a LOOK policy, the arm sweeps across cylinders in one direction, then in the reverse direction when there are no more request to service ahead of it.

Assuming that requests are independent and uniformly distributed over all cylinders and approximating linearly the arm transfer function, it can be shown ([11], [5]) that :

$$E\,[Tseek] < S_m + \frac{S_M - S_m}{1.5\ (L+1)} \qquad \text{when } L \simeq 1$$

and

$$E\,[Tseek] \simeq S_m + \frac{S_M - S_m}{1.72 \times L} \qquad \text{when } L >> 1$$

with - L = number of requests waiting for the arm (including the one which is being serviced)

- S_m = seek time to skip one cylinder
- S_M = seek time to skip all cylinders

The service stations in the queueing network model are (see Fig. 3) :

- Station 1,2 : Arm servers of modules 1 and 2
- Station 3 : Channel, common to both modules
- Station 4 : CPU of File-Processor ; represents only the part of CPU treatment which is serial with channel transfer.

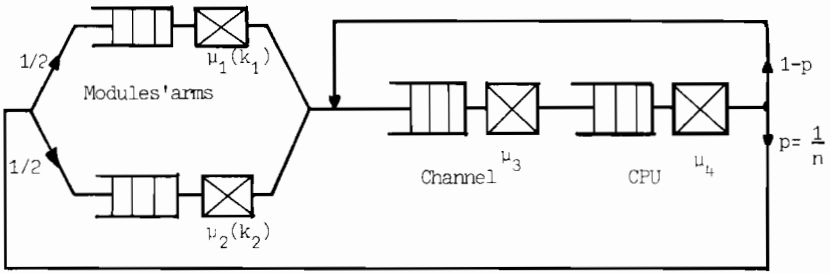


Figure 3 : File Processor model

Model parameters are :

- n : average number of physical accesses/request
- $\mu_i^{-1}(k_i) = Sm + \frac{S_M - S_m}{1.5(k_i+1)}$; i=1,2

  : average upperbound for arm service time at module i conditioned on the fact that k_i requests are waiting for the arm
- μ_3^{-1}: expected channel service for one access (i.e. expected latency time between two successive blocs, plus transfer time for one bloc, plus T/2n where T is the rotation period of the disks)
- μ_4^{-1}: expected serial CPU service time/access
- K : total number of requests being serviced in the file processor.

We can solve the model using [4] which enables us to obtain the throughput. The response time of the File-Processor to a request can then be obtained using Little's formula. Fig. 4 shows several response time curves, for different values of n, μ_3 and μ_4.

Discussion of the model predictions

The actual value of n will depend on the distribution of requests among different types : most requests should be read, write sequential (n=1,2) or read, write keyed (n=3,4,5). So we may expect that the mean value of n should be between 3 and 4.

An estimation of File-Processor workload, which has to be coherent with our previous estimate for bus-traffic, yields a maximum request input rate no greater than 40 req/sec ([5]). So, from Fig. 4, we see that the File-Processor should never become satured, most requests being serviced in 50-100ms or so.

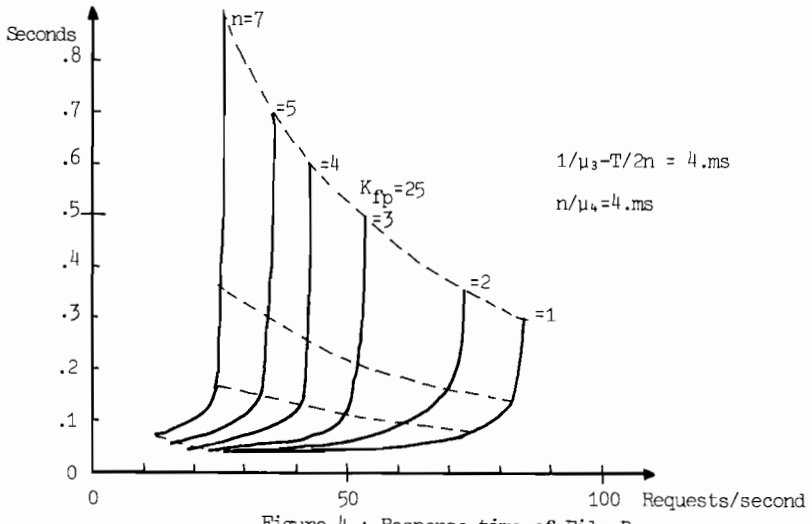


Figure 4 : Response time of File-Processor

The Master-Processor

The Master's functions are :

- interfacing between Executors and terminals (via the bus)
- interfacing between line printers, card readers, etc, and the File-Processor (via the bus)
- allocation of job-steps to Executors
- high level system supervision

Current development of M2 is such that the real complexity of these goals cannot be, even roughly, estimated. So, in the global model proposed below, the Master-Processor does not appear explicity. Instead, we assume only that delays induced through it for terminals and other peripherals are negligible when compared to their own response time. Also, we assume that job-steps will be distributed among Executors in a way such that the workload will be equal for each Executor [10].

The Executor

Executor description

System programs and user's tasks are run on Executors. When a new program (job-step) is activated, it will go through the following steps (see Fig. 2) :

- The Master assigns this program to one of the Executors
- Then the operating system of the Executor will follow one of two paths :
  - if the program is a standard system program, it will be present on the local disk, in its locally executable version, and no further translation is needed
  - otherwise, if it is a user program, the operating system will call its intermediate form in LOC (LOC : Common Object Language) from the File-Processor. Then it will translate it into its own machine language, using its own translator, present on local disk.
- From this point on, the program becomes executable and ready to gain control of the CPU.

At any time, there is only one process in main memory. This process is called the active process, and has control over all ressources belonging to the Executor : CPU, local disk, memory, and also access to the File-Processor. While active, a process can :

- use the CPU to do computations
- send a request to the File-Processor (via the bus), and then wait for an acknowledgment
- do I/O operations on the local disk
- send a message to a terminal (via the bus and the Master).

Executor time is shared through the use of a quantum of maximum resident time (I/O time included), inside main memory : an active process is allowed to stay in main memory for a finite length of time, starting from the end of swap-in and ending at the beginning of the following swap - out. As explained before, a Local Disk will be used, on each Executor, to swap-in and out the content of main memory.

A process will leave the main memory, either when its quantum is exhausted, or when it initiates a new interaction on its associated terminal (or more generally, when a long I/O operation, such as a tape operation, begins). Normal I/O operations on File-Processor disks are supposed to be fast enough so that the CPU of Executor will idle until completion.

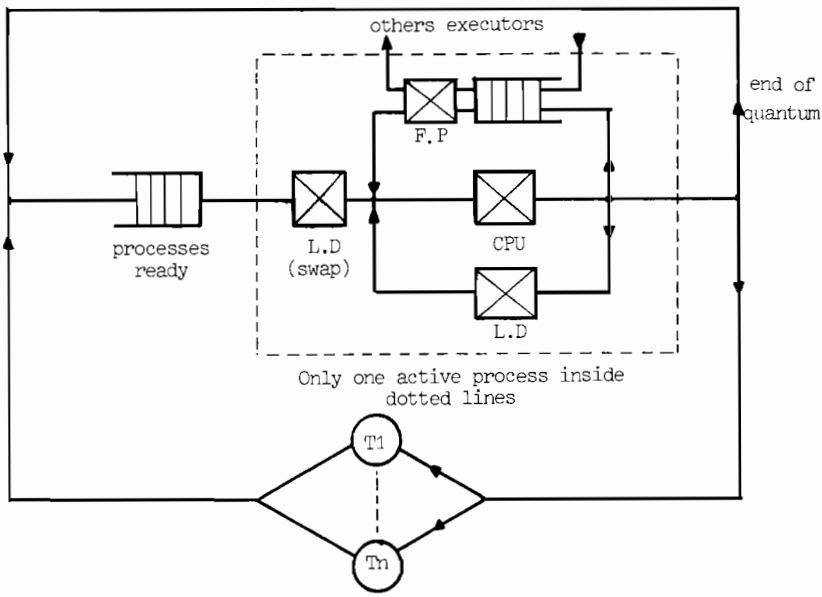
On M2, there will be several values of the quantum, depending on the frequency of interactions between process and terminals. Small values of the quantum will be associated with higher priorities, so that short tasks have a faster response time.

As the scheduling algorithm is not yet determined, the model in its current state deals with a single constant quantum, all tasks being serviced according to a FIFO policy.

There is no need to distinguish batch processes from interactive processes : simply the first ones will make no contribution to terminal activity. The typical workload will result from both types, with a large majority of interactive processes.

The Executor model

It is described on Fig. 5. When a process departs from a terminal, it becomes ready for CPU election. As soon as the CPU is cleared, a new process (if available) is selected, and first, its core image is loaded from the Local Disk (L.D.). Then, the process can either remain active (i.e. let CPU occupied) or leave CPU because of the completion of its quantum, or because of an interaction with a terminal. Executors being monoprogrammed, the process remains active (inside dotted lines of Fig.5) even when waiting an acknowledgment from L.D. or F.P.



L.D : Local Disk

Figure 5 : Executor Model

The previous analysis gave us a low utilisation factor for the bus. This enables us to dismiss the bus waiting time in our model, while including bus service time into other service times, since we can consider that the bus queueing time is negligible. As usual, all service times are approximated by exponential distributions. All processes are supposed to be independent. During a given period of time, a total of Z processes will be present at each Executor, either in think state at one of the terminals, or in system state at the Executor (i.e. in its main memory, or waiting for execution on the local disk).

Model parameters are :

- Z : number of processes waiting at the Executor, including the one in execution
- R : expected think time at any terminal (including delays at the Bus and Master)
- F=F(i) : expected response time of the File-Processor (including bus delay), when it is processing a given number, i, of requests.
- Q : maximum resident time in main memory, including I/O time, for a task during any one passage. It is also the maximum time between the end of swap-in and the beginning of swap-out and acts as a "quantum" of executor time allocation
- C : expected CPU time consumption, for any process, between two successive terminal interactions
- I : expected CPU time between two successive requests to the File-Processor
- E : expected CPU time between two successive accesses to the Local Disk.
- S : expected total swap time (swap-in + swap-out)
- D : expected service time for a local disk access.

Remark 1

L.D (Local Disk) appears twice in the previous model (see Fig. 5). This is uncommon, but not a problem because there can be only one process active inside the dotted lines on Fig. 5, thus no conflict occurs at the L.D. Using this artificial decomposition, it is then possible to have two different service times, S and D, depending on the current state of the process (the same result could have been achieved using two different classes for customers). S represents the service time of the L.D for a swap, while D is used for a local I/O operation during process execution.

Remark 2

Executors are mono-programmed. Therefore any request addressed to the File-Processor or L.D access will leave the CPU idle until completion. Consequently, if X Executors are connected to the bus, at any time, no more than X requests will be at the File-Processor, waiting for service.

Remark 3

Swap-in and swap-out time are summed into S because usually the first one is much greater than the second. Therefore, the distinction between the two would not enhance very much the accuracy of the model, althrough introducing greater complexity into it.

The mathematical model

The following approach is similar to one which has been used for virtual memory analysis in [1][3]. The main differences are that a resident time quantum exists instead of a CPU time quantum and that the model represents mono-programming instead of multi-programming.

The successive solution steps are presented on Fig. 6 (see next page) ; they are needed to solve the Executor model which was presented on Fig. 5.

To begin with, we need to relate elapsed time in main memory to CPU time and I/O response time. Then the quantum will be introduced. Starting at the CPU level, we first examine what is going on inside the dotted lines of Fig. 5, when there is an active process. We assume that all CPU interrupts are of null duration and that the quantum is infinite ; thus we obtain a "virtual" CPU service rate (Fig. 6a), which is the sum of all independent CPU interrupt rates :

$$\alpha = \frac{1}{E} + \frac{1}{C} + \frac{1}{I}$$

Branching probabilities at the output of CPU should be so that departure rates towards L.D, terminals and F.P are respected. (Also, as the probabilities are constants, all service rates should be exponentially distributed for the following resolution method to stand). If α is the departure rate (interrupt rate) from CPU, and a_1 is the static probability to go to L.D when leaving CPU, $a_1.\alpha$ is therefore the I/O request rate for L.D. But this rate is assumed to be equal to 1/E, which is an intrinsic characteristic of the program. Thus a_1, a_2, a_3 must be so that :

$$a_1 = \frac{1}{\alpha E} \; ; \; a_2 = \frac{1}{\alpha C} \; ; \; a_3 = \frac{1}{\alpha I} \qquad (a_1 + a_2 + a_3 = 1)$$

As Executors are mono-programmed, the CPU will run idle while an I/O operation is done on L.D or F.P. Knowing that, we obtain the real service time at the CPU, conditioned on the fact that is an active process in main memory, and only one :

$$\hat{\alpha}^{-1} = \alpha^{-1} + a_1 D + a_3 F$$

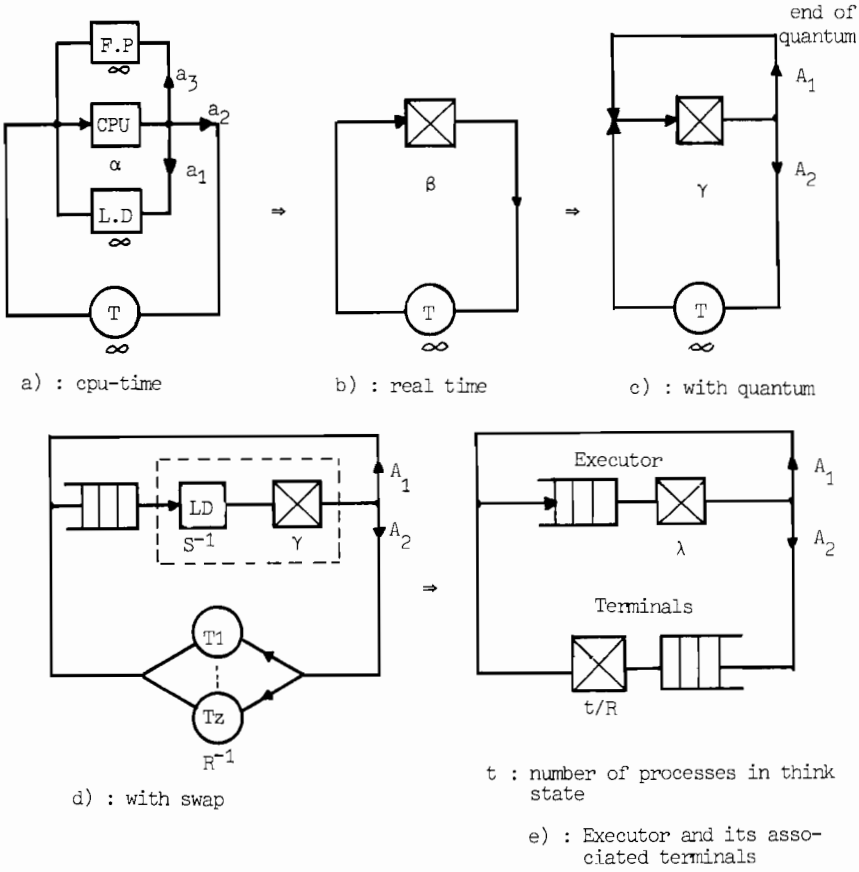


Figure 6 : Resolution steps for Executor Model

At this point, we will use Courtois reduction method [2]. This implies some restrictions on the parameters : their values should be such that interactions between submodels which are being reduced using decomposition occur much less frequently than interactions within the submodels. Assuming it is so, we aggregate F.P CPU and L.D (Fig. 6a) into a single equivalent server of service rate (Fig. 6b) :

$$\beta = a_2.\hat{\alpha} \quad .$$

Up to this point, the quantum was supposed to be infinite. If it is not so, assuming that β is exponentially distributed, we can deduce the probability of needing more than one quantum between two terminal interactions. It is :

$$A_1 = e^{-\beta Q} \qquad (A_2 = 1-A_1)$$

β being exponentially distributed, A_1 is also the probability of needing n+1 quantums, or more, knowing that we used n before.

Thus A_1 is also the probability for a process to leave main memory because of quantum end interrupts.
Owing to quantum interruptions, the service rate at main memory becomes (Fig. 6c) :

$$\gamma = \frac{\beta}{1-A_1}$$

This is so because, when distributions are exponential, the quantum has no effect on throughput (as long as no overhead is implied).

Reintroducing swap service time, we finally obtain the service rate of an equivalent Executor (Fig. 6e) :

$$\delta = [\gamma^{-1} + S]^{-1}$$

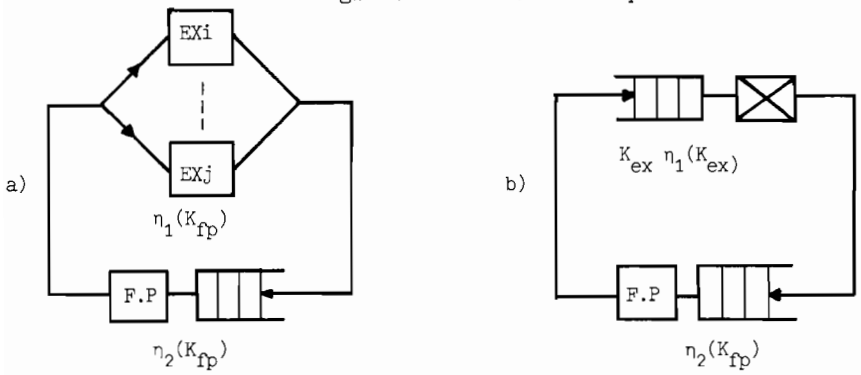
As this value was obtained for a given value of F, i.e. F(i), it is conditionned on i requests being present inside the F.P.

The Global Model

By merging the F.P model and the Executor model into a unique global model, we will obtain global results, such as mean response time of M2.
To do this, we need to take into account all interactions between the F.P and the Executors. These interactions are in fact exchanges of requests between Executors which produce them, and the F.P which services them.

As Executors are mono-programmed and so idle until completion of requests, the mamimum number of requests at the F.P is equal to the effective number of Executors connected to the Bus ; this is true because most programs will not use simultaneity for I/O operations on F.P, and because, as response time of F.P is short, most requests sent to F.P by a program will return before the program is swapped out at the end of its quantum. Thus, we will use a specific model for request circulation (Fig. 7a), with a fixed number of customers (requests) circulating through it. This model will give us the probability P(i) of finding i requests in service at F.P. Here again we use implicity Courtois' reduction method [2] since F.P and Executors are supposed to be stationary between two interactions.

Figure 7 : File-Processor requests model



K_{fp} : number of requests at File-Processor
K_{ex} : number of Executors non-waiting for request
$K_{fp}+K_{ex}=X$: total number of Executors

The equivalent model 7b is solved, using [4], with :

- X = number of Executors connected to the bus
- K_{fp} = number of Executors idling ; also the number of requests at F.P
- $K_{ex} = X - K_{fp}$
- $\eta_2(K_{fp})$ = Service rate at File-Processor, conditioned on the fact that K_{fp} requests are present ; computed from File-Processor model alone (Fig. 3)
- $\eta_1(K_{ex})$ = departure rate of requests from active (non idling) Executors, when K_{ex} of them are active.
  $= K_{ex}(\lambda/(1-\lambda F))$ with $\lambda = \delta\rho A_2 \frac{a_3}{a_2}$ computed for $F = F(K_{fp}) = F(X-K_{ex})$

  where ρ is the utilisation factor of an Executor, as obtained when solving Executor model alone (Fig. 6e).
  Here we assume that, although all Executors may be different, they will have similar workloads, and contribute for equal parts to F.P workload. This hypothesis reflects only present situation, and could be released without too much difficulty.

Solution for model 7b gives us the true distribution of requests among File-Processor and Executors :

$$P(i) = \text{Probability [number of requests at F.P = i]}$$

Knowing P(i), we can obtain, for instance, system M2 global response time. In order to do so, we first compute all values of response time, associated with all possible values of K_{fp} ; using Executor model alone (Fig. 6c), we obtain

$$Trep(i),\ i = 0 \text{ to } X$$

(Response time is the time spent in system state between two successive interactions at a terminal).
Then, we solve the requests model (7b) and find P(i).
Finally, we obtain :

$$Trep = \sum_{i=o}^{X} Trep(i)\ P(i)$$

Trep is the expectation of global response time, and does not depend any more on the state of the F.P (see. Fig. 8).

Validation of the model

This model is intended to represent a real computing system. So validation will be done using measurements on the real system. Unfortunately, M2 will only be fully operational at the end of 1979. Comparisons with a simulation model can be found in [6].
Although not yet validated, the present model can be very helpful to evaluate different design options such as the number of tasks per Executor, or the quantum size, and to determine at what level of load the common ressources of the system will be satured.

Global Results - Conclusions

Values for expected response time of M2 are given on Fig. 8, for several kinds of program behaviour (parameter C) and different values of Z (number of active processes on each Executor).

We see that, if our curves are typical enough of real workload, to achieve a mean response time of 1s or so, a maximum of 6 to 15 tasks can be allocated to each Executor.
Consequently, the L.D being very slow, the overhead ratio will be very high : typically 50% or more.

On Fig. 9, we find mean response time of the File-Processor model alone : saturation at the File-Processor will never occur in the present configuration.

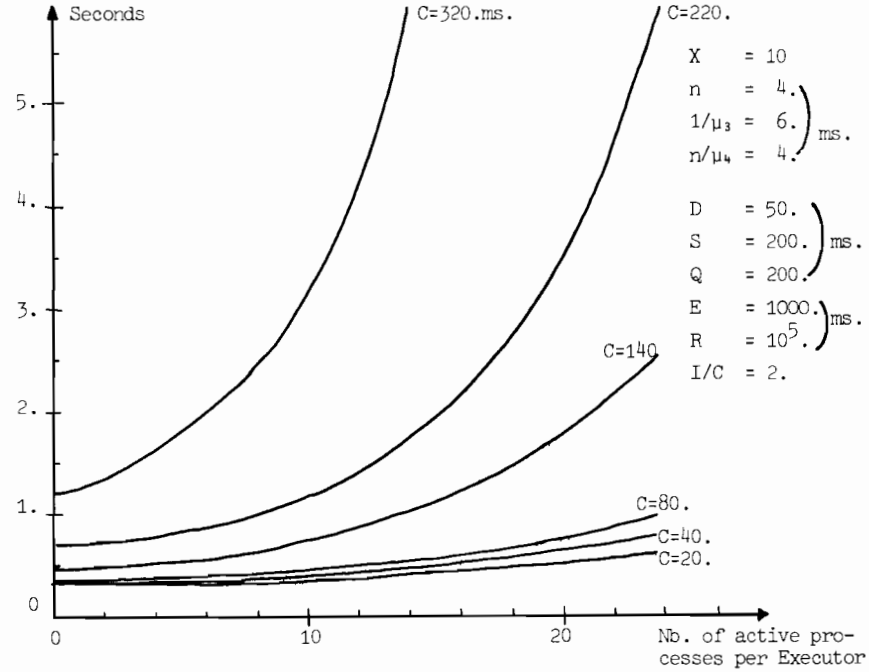


Figure 8 : Global response time of M2.

Parameters : same as on Figure 8.

C	F_{mean} (ms)
20	54.81
40	54.81
80	54.80
140	54.77
220	54.74
320	54.69
440	54.63

for 60 active terminals (6 times 10), bus delay included (6.ms)

Figure 9 : Values of F_{mean}, Global Load

Acknowledgments

I want to thank very much Professor Gelenbe at Paris, Orsay University, Professor Hebenstreit at E.S.E. and the whole development team of M2 at E.S.E. for giving me the opportunity to do this work, and whose great help was essential to its achievement.

References

[1] A. Brandwajn, J. Buzen, E. Gelenbe and D. Potier (1974) : A model of performance for virtual memory systems. Proc. ACM - SYGMETRICS SYMP., Montreal.
[2] P.J. Courtois (1971) : On the near complete decomposability of networks of queues and of stochastic models of multi-programming systems. Carnegie Mellon University, Dept. Computer Science, Res. Rep.
[3] E. Gelenbe, A. Kurinckx (1978) : Random injection control of multiprogramming in virtual memory. IEEE Transactions on Software Enginnering.
[4] W.J. Gordon and G.F. Newell (1967) : Closed queueing systems with exponential servers. Oper. Research, 15, p. 254-265.
[5] J.J. Guillemaud (1978) : Modèle M2 - Résultats partiels. Rapport Interne E.S.E.
[6] J.J. Guillemaud : Modèle M2. Thèse de Docteur Ingénieur, Rapport Interne E.S.E. à paraitre.
[7] J. Hebenstreit (1978) : Project M2 - A multi-mini data processing system. Proc. Eurocomp., p. 431-441.
[8] C. Ihermitte, J.P. Szylowicz (1977) : Etude et réalisation d'un système de gestion de fichiers pour un système multi-minis-ordinateurs. Rapport E.S.E.
[9] A.G. Merten (1970) : Some quantitative techniques for files organization. Ph.D. Th. - Tech. Rep. 15, University of Wisconsin, Computer Center.
[10] Y. Noyelle (1978) : Projet de spécifications du système M2. Rapport Interne E.S.E.
[11] T.J. Teorey and T.B. Pinkerton (1972) : A comparative analysis of disk scheduling policies. Congress of ACM, vol. 15-3, p. 177-184.

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

REGIME PROCESS ANALYSIS OF A VIRTUAL MACHINE OPERATING SYSTEM

Wen-Te K. Lin*
Sperry Research Center
Sudbury, Mass. 01776

A new method of data analysis - stochastic regime process - is applied to a set of data collected from a virtual machine operating system. Some interesting behavior of the system which is independent of the regime-behavior of the system is observed. Although the method is applied to an operating system, it can be used for other software trace data to discover some intrinsic non-obvious characteristics of the system, and validate statistical assumption of stationarity which we often need in doing statistical modeling.

INTRODUCTION

Is it possible to model some activities of an operating system by a stochastic process or by other statistical methods? The answer depends heavily on whether these activities possess the property of independence or stationarity, because many of the available statistical methods require these assumptions. Very often a system analyst simply makes these assumptions without investigating whether they are valid or not. Therefore the results can be either unsatisfactory or misleading. This paper discusses the data collected from CP/67 operating system, and investigates some activities of the operating system to see whether they exhibit characteristics of a stationary or regime process and how these activities correlate to each other. Although there are a few other papers which analyze the same system [1] [2] [3], this paper takes an entirely different approach from a regime process point of view [4] [5] [6] [7].

SYSTEM LOAD AND ITS REGIME-LIKE BEHAVIOR

The system, from which we collect the data, is CP/67 running with batch MVT and more than 40 CMS terminal users in a university environment. CP/67 runs in either CP-mode (supervisor-mode), problem-mode (program-mode), or idle-mode. Fifteen minutes of data was collected which recorded the times when the system went into or out of each of these three modes. It consists of several hundred thousand items. The data were divided into 180 units of 5 seconds each.

Let Pi, Ii be the lengths of time CP-67 spends in problem-mode and idle-mode, respectively, in the ith interval. Let us consider the Pi's and Ii's, for i=1,2,... 180, as time series. These two time series are plotted against time as in figures 1 and 2 for i from 1 to 100.

From figures 1 and 2 we can see intuitively that they are more likely to be regime processes than stationary processes [7]. The first ten points constitute a regime, the next 40 points another. The third regime is from the 51st to the 61st point, the last regime from the 62nd to the last point. We can see that the first and the third regimes have similar averages and variances, as do the second and the fourth. Since, in some sense, problem-time represents the load of the system, we can say the load of the system varies like a 'regime process' with

*This paper is part of the Ph.D. thesis of the author done at Brown University, Providence, R.I. under supervision of Professor Ulf Grenander.

This work was supported by the Information Systems Program of the Office of Naval Research under Contract N00014-75-C-0461 with Brown University.

each regime consisting of independent, identically distributed random variables. This claim is made intuitively without any mathematical support, but one can formulate a regime model for such data and analyze it accordingly [4] [5].

CP-TIME AND PROBLEM-TIME EMPIRICAL FREQUENCY FUNCTIONS

Even though the system load looks like a regime process, the behavior of other parts of the system looks more like that of a stationary process and does not seem to be influenced by the load at all.

Let X1 be a random variable which is the duration CP/67 stays in CP-mode without being interrupted by problem-mode or idle-mode.

Let X2 be a random variable which is the duration of time the system stays in problem-mode without being interrupted by CP-mode or idle-mode.

Let X3 be a random variable which is the duration of time the system spends in idle-mode without being interrupted by CP-mode or problem-mode.

Now let us take the samples of X1, X2, X3 from the first minute portion of the data, and plot their empirical frequency functions, as shown in figure 3, figure 5, and figure 7; similarly for the second minute portion of the data as shown in figure 4, figure 6 and figure 8.

From figure 3 and figure 4 we can see that the CP-time empirical frequency function does not change much even though loads of these two one-minute intervals are so different as discussed in the last section. From figure 5 and figure 6 we can see also that the problem-time empirical frequency function is not affected by the load change. From our knowledge of CP/67 the first observation may be a characteristic of the system, and the second observation is probably the result of the fact that the system is a time-sharing system with an interrupt mechanism. For example, the second property may not appear in the CDC-6600 system because the system does not use interrupts to handle I/O channels. As for figure 7 and figure 8, it is understandable that they look so different because X3 (idle-time) certainly is dependent on the load of the system. The usual Chi-square test fails to confirm that figures 3, 4 and figures 5, 6 come from the same frequency distribution. The test is very sensitive to small values; when these small values are eliminated, the pairs of empirical frequency functions are found to be not significantly different at 10% level.

In Tables T1, T2, T3 we tabulate the empirical means and variance for X1, X2, and X3 for seven consecutive two-minute data sets, and we can see that there is not much change in either the mean or variance in these seven different data sets, even though they correspond to periods of different system loads as can be seen from figure 1 and figure 2.

	1	2	3	4	5	6	7
MEAN	0.79	0.73	0.68	0.63	0.78	0.81	0.78
S.D.	1.00	0.84	0.80	0.70	0.85	0.89	0.93
NO. OF OBSERVATIONS	15432	19823	41443	19302	21903	16712	19734
LENGTH OF OBSERVATION TIME (in minutes)	2	2	2	2	2	2	2

TABLE T1

CP-time empirical frequency distribution table
(Mean and S.D. in units of millisecond)

	1	2	3	4	5	6	7
MEAN	4.54	5.28	5.11	5.53	4.60	4.58	3.99
S.D.	10.53	10.21	10.32	11.63	8.67	8.75	8.67
NO. OF OBSERVATIONS	13058	19442	38367	18994	21340	14642	17148
LENGTH OF OBSERVATION TIME (in minutes)	2	2	2	2	2	2	2

TABLE T2
Problem-time empirical frequency distribution table
(Mean and S.D. in units of millisecond)

	1	2	3	4	5	6	7
MEAN	20.44	7.76	8.91	9.06	8.28	19.33	13.98
S.D.	61.04	5.53	8.03	6.64	6.35	42.22	25.45
NO. OF OBSERVATIONS	2374	381	3076	308	563	2070	2586
LENGTH OF OBSERVATION TIME (in minutes)	2	2	2	2	2	2	2

TABLE T3
Idle-time empirical frequency distribution table
(Mean and S.D. in units of millisecond)

A STATISTICAL INFERENCE METHOD FOR DETECTING MARKOV REGIMES

By a Markov regime process, we mean a stochastic process which exhibits stationary Markov property within certain time intervals, but changes its characteristics between different time intervals. For a more vigorous definition of regime process, see [7]. A statistical inference method has been developed by the author which can be used to detect the transition points between regimes [6]. After these transition points have been found, data within each regime can be considered as observations from a stationary Markov process, and can be analyzed accordingly. This inference method will be described in this section without proof.

Assume there are two samples from a discrete time Markov regime process with n states, each of the sample has T and S observations respectively. Let $N=(n_{ij})$ and $M=(m_{ij})$ be the corresponding transition matrices, i.e. n_{ij} is the number of observations in the first sample which transit from state i to state j. Let t, s, and d be vectors defined as follows:

$$t = (n_{11},\ldots,n_{1n}, n_{21},\ldots,n_{2n},\ldots,n_{n1},\ldots,n_{nn})$$

$$s = (m_{11},\ldots,m_{1n}, m_{21},\ldots,m_{2n},\ldots,m_{n1},\ldots m_{nn})$$

$$d = \sqrt{T}\ (t/T - s/S).$$

Also let C be a n^2 by n^2 matrix defined as follows:

$$C = \text{Covariance}\ (d, d^t)$$

Where d^T is the transpose of vector d, and entry c_{ij} of C is equal to the convariance of the ith element of d and jth element of d^T.

The following theorem will be stated without proof.

Theorem: If the two Markov chain samples are uncorrelated, and come from the same regime of the Markov chain, and if matrix C is positive definite, then

$$Q = dC^{-1} d^T$$

has a chi-square distribution with n^2 degree of freedom.

MARKOVIAN REGIME BEHAVIOR OF THE SYSTEM

Here we will examine the system at a more detailed level from a Markov regime process point of view.

CP/67 is composed of more than 70 modules, each of which handles a specific function. We grouped these 70 modules into 12 larger modules, and collected data which recorded the times when the system transited from one module to the other. The data are 18 minutes long, and consist of several million items. The statistical inference method mentioned above was then used to analyze the data. It confirmed that the system behaved as a Markov regime process rather than a homogeous stationary Markov process in those 18 minutes. It consists of several regimes each ranging from 1 second to 15 seconds in length. The first two regimes were then picked for further examination.

Let T_{ij} be the length of time CP/67 spends in the ith module before going to the jth module, and let T_i be the length of time CP/67 spends in the ith module before going to any other module. Then some of the histograms (empirical frequency distributions) for T_{ij}'s and T_i's are plotted in graphs from figure 9 to figure 15.

From figure 9 and figure 10, we can see that there are more long idle periods in the first regime than in the second regime, therefore the load in the second regime is heavier than in the first regime. From figure 11 and figure 12, we can see that the second regime has twice as much paging activity as the first regime (the scale of Y axis in figure 12 is twice as large as in figure 11). From figure 13 and figure 14, we can see that the first regime has twice as many privileged operations as the second regime. Hence we can safely conclude that these two regimes are very different in their operational characteristics; the second regime is more I/O bound than the first one. Nonetheless, the shapes of all the graphs for T_{ij}'s and T_i's in these two different regimes look strikingly similar. Based on these observations, we believe that the invariance of the distributions of T_{ij}'s and T_i's in these two different regimes may be a characteristic property of CP/67.

CONCLUSION

This study demonstrates the usefulness of regime process analysis technique in analyzing program behavior. It exposes the underlying regime process behavior of a stochastic process. By doing so, it enables us to use more conventional statistical tools within each regime.

By applying this technique to the data collected from CP/67, we have shown that the operating system CP/67 behaves as a Markov regime process. The characteristics of some CP/67 system parameters are shown to vary between regimes, while others are shown to be invariant between different regimes.

Therefore before we make any statistical assumptions about a system when we do system modeling, it is useful that we collect data from the system, if possible, and do a regime process analysis. Then we will more likely be able to avoid building a model that cannot be validated.

REFERENCES

[1] Y. Bard & K. V. Saryanarayana, "On the Structure of CP/67 Overhead", Statistical Computer Performance Evaluation, Academic Press, 1972.

[2] Y. Bard, "Experimental Evaluation of System Performance", IBM System Journal, Vol. 12, No. 3, 1973.

[3] Y. Bard, "An Analytic Model of CP/67 and VM/370", Computer Architectures & Networks, edited by E. Gelenbe, American Elsevier Publishing Co., New York, 1974.

[4] Ang, Bent-Tune, "A Heuvistic-Adaptive Procedure for Segmentation of Time Patterns", International Journal of Computer and Information Science, Vol. 4, No. 4, 1975.

[5] Grenander, U., etc., "Patterns in Program References", IBM Journal of Research and Development, May 1975.

[6] Lin, W. T., "A Statistical Study of CP/67 Operating System" Ph.D. Thesis, Brown University, 1974.

[7] Adenstedt, R.K., "Weather Regimes in Stochastic Meteorological Models", Quarterly of Applied Math. 28, 343 (1970).

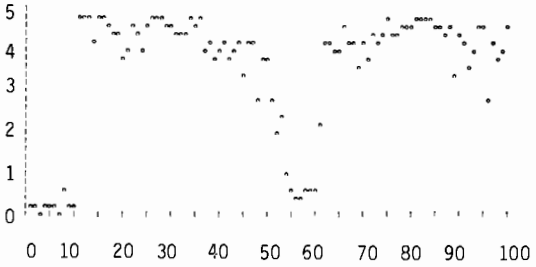


Figure 1.
Let PRi be the length of time (in second) CP-67 spends in problem-mode in the i's interval (where each interval is 5 seconds), then this graph is the plot for PRi where i=1,2,...,100.

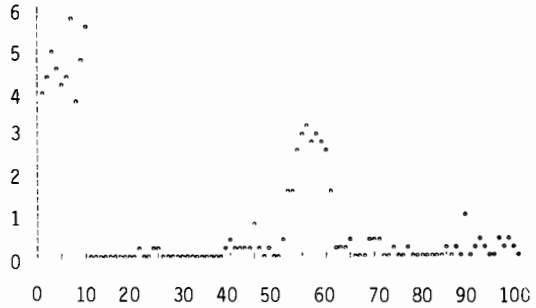


Figure 2.
Let IDi be the length of time (in second) CP-67 spends in Idle-mode in the i's interval (where each interval is 5 seconds), then this graph is the plot for IDi where i=1,2,...,100.

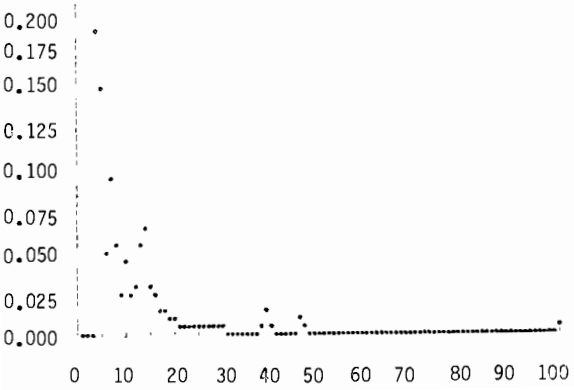


Figure 3.
Let X1 be a random variable which is the duration of time the system stays in CP-mode without being interrupted by Problem-mode or Idle-mode (the unit is in timer unit), then this graph is the empirical frequency distribution of X1 for the first minute portion of the CP/67 data.

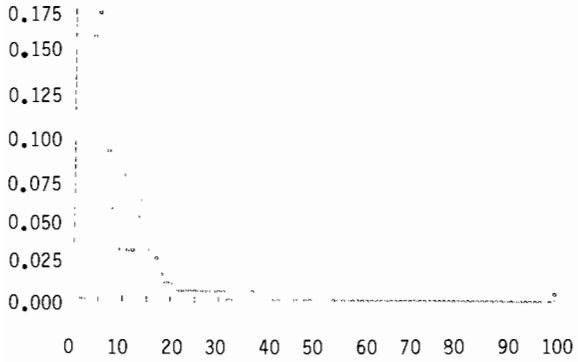


Figure 4.

Let X1 be a random variable which is the duration of time the system stays in CP-mode without being interrupted by Problem-mode or Idle-mode (the unit is in timer unit), then this graph is the empirical frequency distribution of X1 for the second minute portion of the CP/67 data.

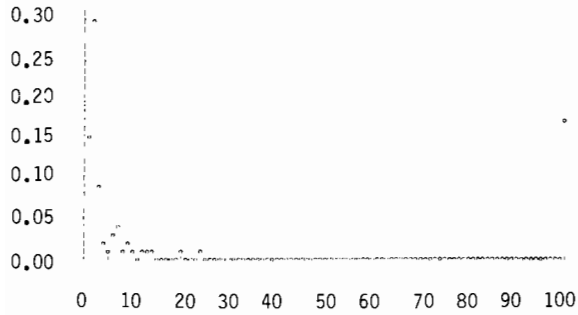


Figure 5.

Let X2 be a random variable which is the duration of time the system stays in Problem-mode without being interrupted by CP-mode or Idle-mode (the unit is in timer unit), then this graph is the empirical frequency distribution of X2 for the first minute portion of the CP/67 data.

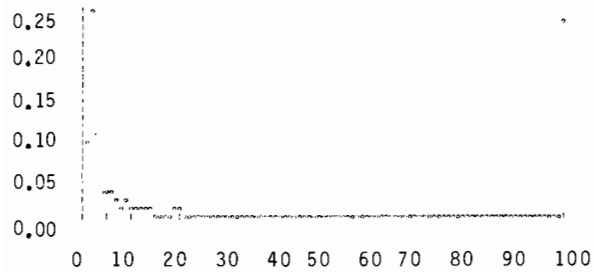


Figure 6.

Let X2 be a random variable which is the duration of time the system stays in Problem-mode without being interrupted by CP-mode or Idle-mode (the unit is in timer unit), then this graph is the empirical frequency distribution of X2 for the second minute portion of the CP/67 data.

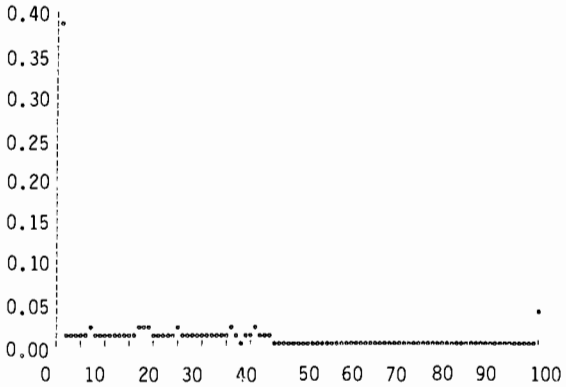


Figure 7

Let X3 be a random variable which is the duration of time the system stays in Idle-mode without being interrupted by CP-mode or problem-mode (the unit is in timer unit), then this graph is the empirical frequency distribution of X3 for the first minute portion of the CP/67 data.

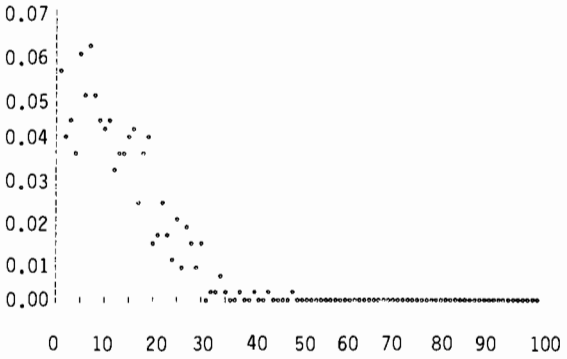


Figure 8

Let X3 be a random variable which is the duration of time the system stays in Idle-mode without being interrupted by CP-mode or Problem-mode (the unite is in timer unit), then this graph is the empirical frequency distribution of X3 for the second minute portion of the CP/67 data.

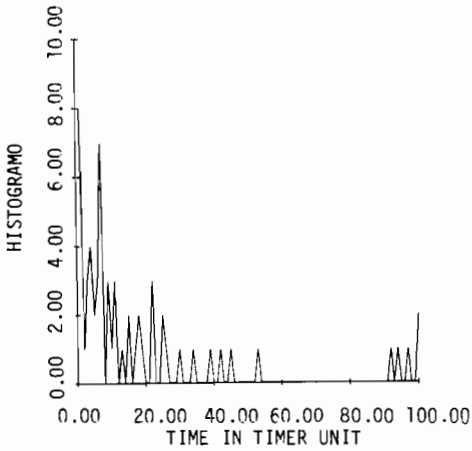


Figure 9

Let Ti be the length of time CP/67 spends in ith module before going to any other module. Then this graph is the empirical frequency distribution for Ti in the first regime with the ith module being the Idle-state.

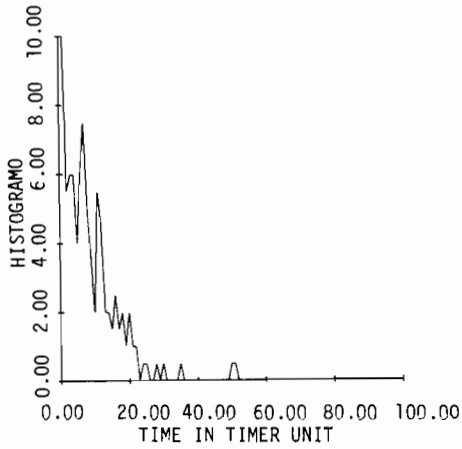


Figure 10

Let Ti be the length of time CP/67 spends in ith module before going to any other module. Then this graph is the empirical frequency distribution for Ti in the second regime with the ith module being the Idle-state.

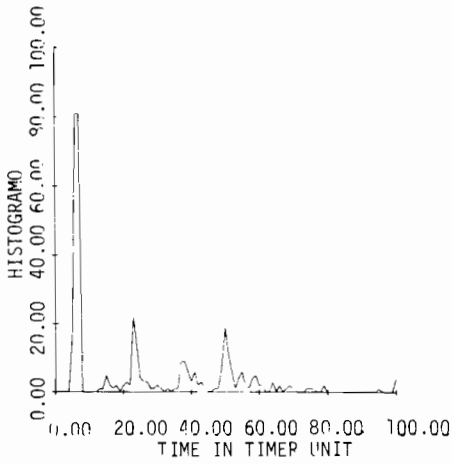


Figure 11

Let Ti be the length of time CP/67 spends in ith module before going to any other module. Then this graph is the empirical frequency distribution for Ti in the first regime with the ith module being the Paging-module.

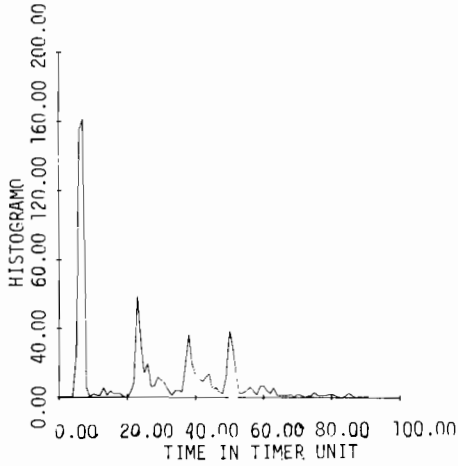


Figure 12

Let Ti be the length of time CP/67 spends in ith module before going to any other module. Then this graph is the empirical frequency distribution for Ti in the second regime with the ith module being the Paging-module.

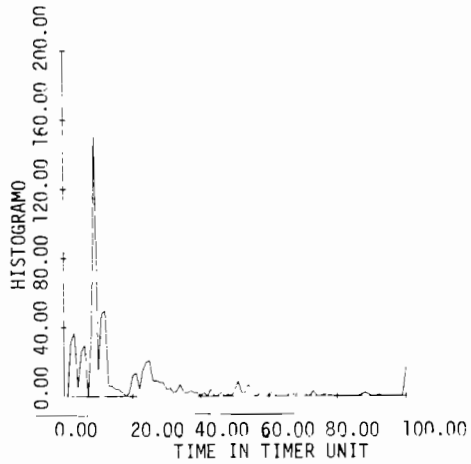


Figure 13

Let Ti be the length of time CP/67 spends in ith module before going to any other module. Then this graph is the empirical frequency distribution for Ti in the first regime with the ith module being the Privileged-operation-module.

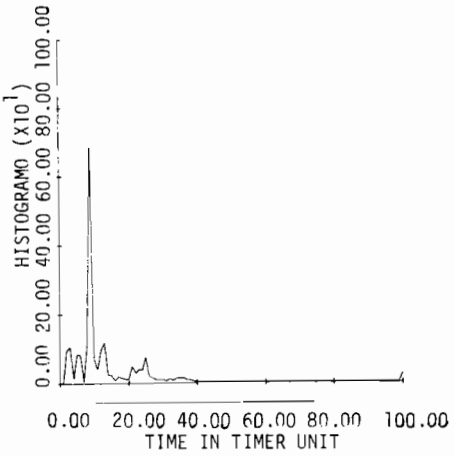


Figure 14

Let Ti be the length of time CP/67 spends in ith module before going to any other module. Then this graph is the empirical frequency distribution for Ti in the second regime with the ith module being the Privileged-operation-module.

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

A HYBRID SIMULATION /ANALYTICAL MODEL OF A BATCH COMPUTER SYSTEM

D. ASZTALOS

Computer Center of Hungarian Planning Office. Budapest,Hungary

The presented model consists of two parts: a trace-driven simulation part realizing the dynamic control of CPU priority allocation, and a simple closed queueing network model with two servers /CPU and I/O device/. The service discipline at the CPU is a priority driven preemptiv-resume one, while the I/O device is of infinite capacity. The model is calibrated and validated using the method presented in [8] .

INTRODUCTION

The rapid evolution of modelling techniques makes possible to construct rather complex models of computer systems. The most powerful tools developed in the recent years are the multi-class local balance queueing network model and its solution techniques[3,4,10]; the decomposition technique 6 , well formalized simulation techniques [12]. The statistical methods for calibration and validation of the models are developed in less extent; there are very few papers considering this aspect of the modelling practice [5,8].

Our aim was to develop a simple model of a batch computer system which has a stronger structural relationship with the real system than that presented in [8]. We hoped that this model will provide more accurate estimates of the elapsed time, and the method of calibration and validation of [8] was used to prove that our assumption is true.

The modelled system is presented in Section 1. The model and the practical considerations which led to select this model are discussed in Section 2. The calibration and validation of the model are given in Section 3.

1. THE MODELLED SYSTEM

The modelled system is a medium size ICL System 4/70 configuration operated in batch environment. It consists of a central processor unit, a disc subsystem with ten disc drives on three channels /off-line seek and dual channel options are provided/, twelve magnetic tape units, usual slow peripherals and 704 Kbytes of core store.

The functional diagram of the modelled batch system is shown on Fig.1. The main parts of the system are the job queue, the scheduler, the allocator of permanent resources and the execution cycle. The jobs arrive from the central card reader. Each job arriving into the system enters the job queue and activates the scheduler which checks whether the permanent resource requests of the job can be satisfied or not. If all of the requested resources are available then they are allocated to the job till its departure from the system. In other cases the job remains in the job queue. Each job leaving the execution cycle activates the scheduler and the permanent resources of the job are deallocated.

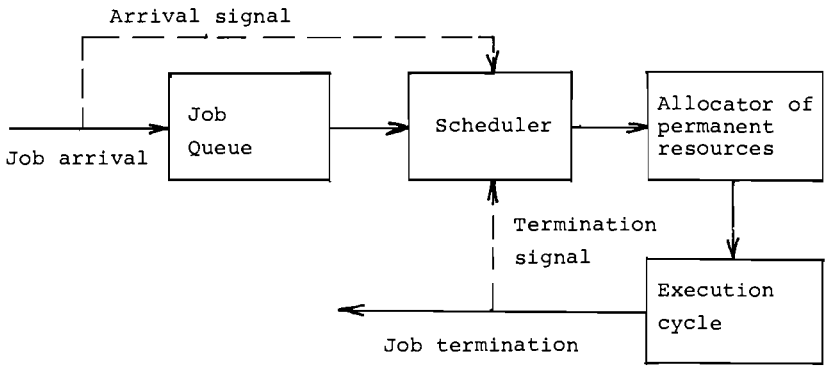


Figure 1. Functional diagram of the modelled system.

In this case the scheduler tries to find one ore more jobs in the queue whose requests for permanent resouerces can be satisfied. The events involving the activation of the scheduler /job arrival and job departure/ are the so called scheduling events. In our system

there are two types of permanent resources: core store and tape units. The handling of the core store is static, i.e. to each job is allocated one continuous part of the core store, the size of which is defined by the job request. There are no swapping and garbage collection options.

Any job may require one or more tapes to be mounted before entering the execution cycle.

The execution cycle contains the following hardware resources: CPU, channels and peripherals. Usually there are more than one job running in the execution cycle in multiprogramming mode, so there are special data constructs contained in the core store: the CPU-, channel-, and device-queues. The execution of any job is the service of its cycling CPU and I/O requests, however there are jobs whose computing and transfer activities partially may be overlaped. The service disciplines /dispatching algorithms/ applied to the CPU-, and device-queues are based on dynamic priority rules. In the CPU-queue the job which has used the most CPU time for the last Δt time interval has the lowest priority during the next Δt time interval. A preemptive-resume priority rule is applied at the CPU queue. The disc dispatching algorithm has a head movement optimization feature keeping the movement of the head in one direction as long as possible. Some aspects of the above computer system have been measured and evaluated in the recent years using a software monitor [1,2]. Some assumptions of the model described in the next section are based on these measurement results.

THE MODEL AND RELATED CONCEPTS

The model has been created as a trace-driven simulator. During normal batch operations, data describing certain job characteristics /such as arrival time and resource requirements/ are collected in the System Journal which is later processed to form the so-called trace, the main input for the simulator. The collection of trace data is accompanied by the collection of data about the performance of the real world. These real-world data are used during calibration as the desired simulator output with which the actual simulator output should provide a good match. We should mention that our first aim was to find out with which level of accuracy can be modelled execution cycle of our system using a relatively simple analytic

model.This is the reason why the job queue and tape mount handling were excluded from the model. The simulation of these parts of the system will be included in the next phase of the research.

The job characteristics used in the model and giving one record of trace data are the following:

1. CPU time in problem state
2. numer of transfers
3. arrival time of the next job relative to the start of the current one
4. measured elapsed time of the job.

The job elapsed time is the time from when a job enters the execution cycle to the time the job leaves it. The job execution time is the minimum elapsed time i.e. when the job experiences no competition from other jobs in the execution cycle.

In modelling a system session, the model divides the session up into a series of time intervals. For each time interval the number of jobs /i.e. the level of multiprogramming/ is constant. A time interval is terminated by one of three possible events: job arrival, job termination, or expiration of Δt interval of dynamic CPU priority allocation. The time of the next job arrival t_a /relative to the start of the actual time interval/ is calculated as the difference of arrival time of the next job in the trace data record of the job arrived most recently and the time elapsed from this arrival. The time a job leaves the system is predicted by the analytical model of the execution cycle. At the start of each time interval the analytical model defines the throughput /or execution speed/ v_p of each job in the execution cycle. The throughput is the number of CPU-I/O cycles executed in unit time for a given job. Also, at the start of each time interval, each job in the execution cycle has a remaining execution time t_p, which is given as a real number of remaining CPU-I/O cycles. The job with the minimum t_p/v_p is the job that will terminate first assuming that the CPU priorities are unchanged. The length of the next time interval t_i is given by

$$t_i = \min \left(t_a, \min_p \left(t_p/v_p \right), \Delta t \right)$$

We assume, if there are no jobs in the execution cycle, $\min t_p/v_p = t_a$ Given t_i, the number of executed CPU-I/O cycles for this interval may be computed for each job and it is subtracted

from the value of the remaining number of CPU-I/O cycles $t_p.t_i$ is added to the value of the elapsed time so far for each job except the one with arrival time t_a. This procedure continues until t_p is reduced to zero for a particular job. This represents the time at which the model predicts the job will terminate. The accumulated elapsed time at the simulated time of job termination is then the predicted elapsed time for that job.

The analytical model of the execution cycle used in this experiment is described and solved in[11]. We should note that this model is a particular case of a more general model treated in [9] . The model is a closed queueing network with two servers, the CPU and the I/O subsystem. The service at the CPU is based on a preemptive-resume priority rule. The I/O subsystem is of infinite capacity. Let the number of jobs be n and designate them by the integers 1,2,...,n. We assume that job i CPU service time is exponentially distributed with parameter α_i and each job has the same exponentially distributed I/O service time with parameter λ. Consider now the case when for every i the i-th job has priority over jobs of index higher than i. Denote $\delta^{(i)}$ the CPU busy period length if the number of jobs executed simultaneously is i. Put $\Phi_i(s)=Ee^{-s\delta^{(i)}}$.We shall give a recurrence relation for $\Phi_i(s)$. For this consider a job, say job A, with computation time distributed as $\delta^{(i-1)}$ and I/O time parameter $m_{i-1}=(i-1)\cdot\lambda$. Let B be the i-th job, and suppose that it is executed with job A in a fashion that A has priority over B. Now the CPU busy period resulting from these two jobs is stochastically equivalent to $\delta^{(i)}$. Thus we obtain

$$\Phi_i(s) = \frac{m_{i-1}}{m_i} Ee^{-s\delta_A} + \frac{\lambda}{m_i} Ee^{-s\delta_B}$$

$$= \frac{m_{i-1}}{m_i}\left[\Phi_{i-1}(s+\lambda) + \frac{\alpha_i[\Phi_{i-1}(s)-\Phi_{i-1}(s+\lambda)]}{\alpha_i+s+m_{i-1}(1-\Phi_{i-1}(s))}\right]$$

$$+ \frac{\lambda}{m_i}\frac{\alpha_i}{\alpha_i+s+m_{i-1}(1-\Phi_{i-1}(s))}$$

with $m_i=i\cdot\lambda$, and

$$E\delta^{(i)} = \frac{m_{i-1}}{m_i}\left[E\delta^{(i-1)} + \frac{1}{\alpha_i}\left(1-\Phi_{i-1}(\lambda)\right)\left(1+m_{i-1}E\delta^{(i-1)}\right)\right]$$

$$+ \frac{\lambda}{m_i}\frac{1}{\alpha_i}\left(1+m_{i-1}E\delta^{(i-1)}\right).$$

This formula is a particular case of IV.7.25. in [9] for $N_i=1$, $i=1,\ldots,n$.

The steady-state probatility of empty CPU queue when the number of jobs executed simultaneously is i given by

$$e_i = \left[1 + m_i E\delta^{(i)}\right]^{-1}.$$

The probability that job i is found at I/O subsystem is

$$r_i = \frac{\alpha_i}{\lambda\left(1+(i-1)\cdot\lambda\cdot E\delta^{(i-1)}\right)}\left[1 - \frac{e_i}{e_{i-1}}\right].$$

r_i can be interpreted as the average time spent by the job i in I/O state in each time unit. Thus, the throughput of job i can be calculated as

$$v_i = \lambda \cdot r_i$$

which is interpreted as the average number of CPU-I/O cycles executed for job i in unit time.

Now we shortly discuss those practical considerations which led to the use of the above model.

a/ Infinite capacity of the I/O subsystem is an assumption accepted on the base of measurement rusults [2] . It has been shown that the average channel wait time is less than 1 % of the average time of a transfer request. On the other hand, less than 8 % of the arrivals form a device queue with two or more requsts including the one being serviced. It means that the I/O load is well balanced in the system. Thus, no large error was introduced using the infinite capacity server as the model of the I/O subsystem.

b/ We were forced to use the exponential service time distribution assuption by analytical and numerical limitations. Our formulas contain the Laplace transform of the busy period, which cannot be calculated for general distributions. There is no practical evidence indicating the use of any other types of distributions.

c/ It was very important to include the priority driven CPU service into the model. Using previously the processor sharing service discipline the average absolute error of the estimated elapsed time

was 22 % in the best case.

d/ It is assumed in the model that there is no overlap of the CPU and transfer activities within one job.

In the model, like the reality, each job belongs to separate priority class. They differ only in the value of average CPU service time α_i^{-1} , which is calculated as the ratio of the job's CPU time and the number of transfers, plus an overhead constant. The overhead constant is the same for all jobs and represents the CPU time required by the system to service a transfer /initiation and termination/.

The analytical model has two parameteres /the calibration parameters/ which are set to their final values in the calibration and validation process. They are the overhead constant and the average I/O service rate λ.

The integration of different methods of analysis /simulation, queueing theory/ is possible because the model is decomposed into two parts which communicate through only one type of variables: the execution speed of a job [6] . Further advantages of the hybrid method are discussed in [7] .

3. CALIBRATION AND VALIDATION

Calibration and validation are two stages in the development of a performance model of a computer system during which the degree of confidence in the model is established.

The calibration of the model was an iterative procedure whose objective was to reduce the difference in behaviour between the model and the real system by adjusting the calibration parameters of the model. Calibration was carried out by applying a given trace /Trace 1/ to the model. One method of comparing the difference between the real system and the model is to compare the real job elapsed time with the predicted job elapsed time. The difference between these values is the residual. The absolute value of the residual is used as the 'figure of merit' necessary for deciding wether one version of a model is significantly closer in its predictions to the real word, than another. The nonparametric

Wilcoxon Matched-Pairs Signed-Ranks test was used for this purpose. At the same time, the Method of Good Balance was used to indicate wether the residuals were correlated with any of the job characteristics, e.g. CPU time, number of transfers. The model is well balanced if the error in such a model does not depend significantly on any job characteristics, but instead is randomly distributed amongst all types of jobs. A linear regression analysis was carried out with the residual in job elapsed time as the dependent variable and measures of job characteristics as the independent variables. The objective of the exercise was to develop gradually a regression equation in which none of the regression coefficients are significant. The initial values of the calibration parameters were set using the measurement results of [1, 2] . A selection of the model runs carried out in the calibration phase using Tracel are displayed in Table I.

Table 1. Calibration of the model using Tracel

Run	Oh	λ^{-1}	$\lvert\bar{r}\rvert$	P	MGB
1	2.5	30	42.36	-	NWB
2	3.0	35	35.64	0.0829	NWB
3	3.5	40	33.04	0.1379	NWB
4	4.0	36	28.68	0.0239	NWB
5	4.2	36	25.08	0.0712	WB
6	4.4	36	24.76	0.1477	WB

Mean measured elapsed time t_e = 251.84 seconds

Key: $\lvert\bar{r}\rvert$ mean of absolute va lue of residuals
P probability that there is no difference between Runi and Run i-1
MGB Method of Good Balance
WB Well Balanced model
NWB Not Well Balanced model
Oh overhead constant in milliseconds
λ^{-1} avarage I/O service time in milliseconds

The objective of the validation process is to find a set of parameter values, determined during calibration, with which the model predictions are not significaty different for other traces.

The model parameters were set to values obtained in the calibration of the model with the Tracel. The model was then run with the

Trace2 and Trace3 respectively. A nonparamentric test, the Mann-Whitney U-test, was then carried out to determine if there was any significant difference in the model predictions. The criterion for comparison was the absolute value of residuals. The null hypothesis, that two independent groups of observations have been drawn from the same population, was tested.The independent groups were the sets of residuals obtained by running the model with a given set of parameters, using different input traces. Since there were three traces, the test was carried out in a pairwise manner, comparing two sets of absolute residuals at a time, making three tests in all /see Table II/.

Table II. Pairwise comparison of model predictions Using Mann-Whitney U-test

Session A	Session B	P
Trace I	Trace 2	0.3811
Trace 1	Trace 3	0.2257
Trace 2	Trace 3	0.5637

P, probability of the null hypothesis

Table III. Comparison of modelled sessions

Session	Actual t_e (s)	Predicted t_e (s)	$\|\bar{r}\|$ (s)	$\|\bar{r}\|$ % t_e
Trace 1	251.84	246.44	24.76	9.83
Trace 2	316.26	294.72	32.28	10.20
Trace 3	286.55	271.31	30.12	10.51

Key: t_e mean job elapsed time

$|\bar{r}|$ mean of absolute residuals

The Mann-Whithey test indicates the probability of the null hypothesis, that the two sets of absolute residuals have been drawn from the same distribution, being true. Table II. shows that the null hypothesis cannot be rejected at the 10 per cent level. Hence the model has been successfully validated for the three traces under consideration. Table III compares the predictions of the validated model for the three sessions modelled.

We note that our estimations are significantly more accurate than those presented in [8]. We could achieve this greater accuracy because we used a model which has relatively strong structural relationship with the real system, and those job characteristics in the trace-data which have the greatest influance on the elapsed time of a given job.

4. CONCLUSIONS

A simple hybrid simulation/analytical model of a batch computer system was presented. We claim that the hybrid models have a wide range of applicability. They are the only tools by which, for example, the sophisticated schedulers of complex computer systems can be modelled and evaluated. Another important aspect of our method is that the relevant measurement data should be used as input trace to achieve accurate predictions.

ACKNOWLEDGEMENTS

I am indebted to Professor M. Arato for his invaluable advice and assistance. I am very grateful to G. Spiegel for implementing the simulation program.

REFERENCES

[1] D. Asztalos /1977/ pp.19-27. Performance Evaluation of the Multijob Supervisor. Measuring, Modelling and Evaluating Computer Systems, H. Beilner and E. Gelenbe, /eds./ North-Holland.

[2] D. Asztalos /1977/ Measurement of Disc Activity in the Multijob Time-Sharing System. Proc. of International Seminar on Experiences of Interactrive System Use, Wydawnictwo Politechniki Wroclawskiej, Wroclaw.

[3] G. Balbo, S.C. Bruell and H.D. Schwetman /1977/ Customer Classes and Closed Network Models - a Solution Technique. IFIP 77, North-Holland, 559-564.

[4] F. Baskett, K.M. Chandy, R.R. Muntz and F.G. Palacios /1975/ Open, Closed, and Mixed Networks of Queues with Different Classes of Customers. Journal of ACM, Vol.

22, No.2. 248-260.

[5] H. Beilner and G. Waldbaum /1972/ Statistical Methodology for Calibrating a Trace-Driven Simulator of a Batch Computer System. In Statistical Computer Performance Evaluation, W. Freiberger /ed./, Academic Press.

[6] P.J. Courtois /1977/ Decomposability. Academic Press.

[7] E. Foxley /1978/ A Hybrid Computer Performance Modelling System. Computer Journal, Vol.21, No.3. 205-209.

[8] H. Gomaa /1978/ The Calibration and Validation of a Hybrid Simulation/Regression Model of a Batch Computer System. Software-Practice and Experiment, Vol.8, 11-28.

[9] N.K. Jaiswal /1968/ Priority Queues. Academic Press.

[10] A.E. Krzesinski and P. Teunissen /1977/ Efficient Computational Forms for the Normalising Canstant and the Statistical Measures of Mixed, Multiclass Queueing Networks. Tech. Report RW77-04, Dept. of Computer Science, Univ. of Stellenbosch, 7600 Stellenbosch, South Africa.

[11] J. Tomko /1975/ Processor Utilization Study. Comp. Maths. with Appls. Vol.1, pp. 337-344.

[12] B.P. Zeigler /1976/ Theory of Modelling and Simulation. John Wiley & Sons.

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

AN APPROACH TO THE CONSTRUCTION OF WORKLOAD MODELS

Winfried Materna
Universität Dortmund, Informatik IV
Postfach 500 500
D-4600 Dortmund 50, Germany

Contents

Abstract

This paper presents methods for constructing representative workload models based on statistical analysis and modelling techniques. The real workload is characterized by the magnitude of the demands for various system resources and times at which these demands occur. Work demands and occurence times are described by a non-homogeneous Poisson-process and by a multivariate mixture of densities, respectively. Methods for estimating the different model parameters are presented or developed. The proposed scheme for modelling workloads is exercised on several samples from a timesharing system, which had been collected by a software monitor.

1. Introduction

The evaluation of computer performance requires the specification of the system load on account of the causal relationship between performance and underlying workload. In spite of the high level of different system modelling techniques analysis and modelling of the workload are poorly developed. This work presents a step in the direction of better understanding and handling the load analysis and modelling problem.

Workload is defined as the collection of all jobs (or other load objects) that are processed by a computer system during a specified period of time. The load characterization used here is based on the magnitude of demands for various system resources, like CPU-time- and core requirements, and the occurence times of demands, especially the arrival times of jobs.

Several load models have been devolped, and it appears suitable to classify them according to the evaluation technique they applied:

(i) load models (benchmarks, synthetic programs, traces,...) for experiments on the object system,

(ii) load models (traces, unidimensional empirical or theoretical distributions,.) for simulation experiments,

(iii) load models (unidimensional theoretical distributions) for mathematical

system models.

Load models must be valid, that means the load model must produce the same system behaviour as the real workload for a class of object systems with a specified deviation.

The validity of load models has often be questioned

- the comparison between different load models is difficult or impossible (benchmarks,...),
- the proof of the validity is difficult, and therefore often neglected,
- the models usually postulate the independence of job demands, like execution time, memory requirements, etc. (unidimensional distribution models), which is not true in reality,
- usually there is no description of the arrival process (benchmarks, synthetic programs),
- the models are system dependent.

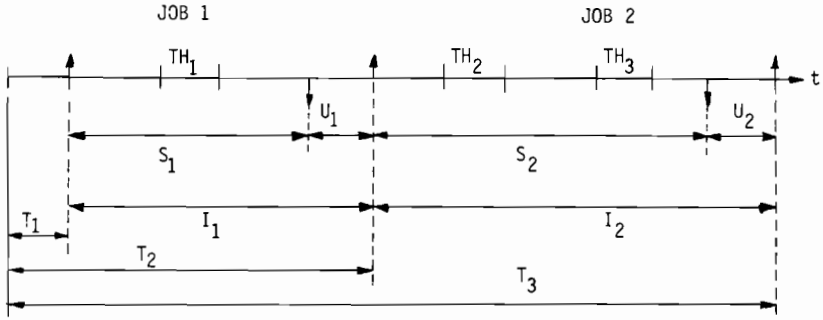
The real workload is considered as a multidimensional stochastic process $(T_j,\vec{X}_j)_{j\in J}, J\subseteq\mathbb{N}$, with the variable T characterizing the arrival time of jobs (or other load objects) and a vector of variables $\vec{X}$ for the job demands. Following this interpretation the model includes the occurence times and work demands generated by a user community, and therefore avoids most of the disadvantages listed above. Comparison of load models is possible due to the statistical nature of the processes.

The modelling techniques developed is based on an extensive system measurement and data analysis. Thus, the practicability of the method can be proven. The experimental data were obtained from a software monitoring system on a DEC-System 1050. 22 samples were recorded, one sample per day, between 9am. - 7p.m., each with a sample size of about 2000 - 4000 jobs.

Only a few approaches in the literature can be compared to the proposed modelling scheme: Clustering models for job demands, e.g. /ARTIS 78/, /AGRAWALA 76/, /FANGMEYER 76/, /LANDAU 76/, and Sreenivasan's and Kleinman's discrete, multivariate distribution model for job demands /SREENIVASAN 74/ (Compare also /MATERNA 78).

2. Arrival Process Analysis and Modelling

The following arrival process analysis and modelling assume the stochastic independence of the time variable T and the vector of demands $\vec{X}$. Thus, it is possible to describe the process $(T_j,\vec{X}_j)_{j\in J}, J\subseteq\mathbb{N}$, by a univariate arrival process model $(T_j)_{j\in J}$ and a multivariate demand model $(\vec{X}_j)_{j\in J}$. The selection of this time variable seems suitable for systems without control of job arrivals; and different statistical methods, like autocorrelation and trend analysis, can be used in an easy way. Furthermore, we get a seperation of variables which exclusively describe job properties $(\vec{X})$ and system or environment properties (T). Different time variables are capable of describing the arrival process. Figure 1 shows some of them. In this approach we will use the arrival time T or the interarrival time $I(I_j,I_j=T_j-T_{j-1})_{j\in J_0}), J_0 = J\setminus\{0\}$.



T_j = job arrival time

I_j = job interarrival time

TH_j = think time of interactive jobs

U_j = time between jobs (sometimes also called think time)

S_j = service time

We are able to construct a

(i) U-model with S-model or
(ii) $TH^*=f(TH,U)$-model
(iii) T-model / I-model.

Figure 1: description of the arrival process

There are different job sources (terminals, card readers,...) at a computer system, each forming a seperate process. In order to simplify the modelling approach all source processes are superposed, so only one process description is needed. Now we get a simpler formal treatment, and the sample size for parameter estimation can be reduced. The analysis of the arrival processe is based on several time series.

There are roughly two situations which arise in the analysis of time series:

(i) Analysis in which specific models are to be tested whereas parameters are to be estimated.
(ii) Analysis in which no particular model is being appeared and the gross features of the data are being examined.

In our situation (ii) the exploratory analysis must be used in order to suggest a pertinent model.
Three steps are particularly important:

(i) descriptive statistical analysis,
(ii) trend analysis,
(iii) autocorrelation analysis.

The first step of descriptive statistical analysis is often underestimated. However, a graphical analysis together with the calculations of moments provide a good insight into the data.

In order to treat the time series or the stochastic process as a stationary one, we have to test the data on any apparent trends. If no trends are found in the data, then it is assumed that the time series is stationary, which implies that the marginal distributions of the I_j's are identical. Since the underlying distribution function is unknown, we apply Spearman's rank correlation coefficient test /SACHS 73/ and Mann's test /HOLLANDER 73/.

Most of the 22 samples analysed have a trend, which indicates an instationary process description. Spearman's test leads to higher values of the statistic in the proved examples (see for example figure 2). A comparison of the power of these tests is unknown.

Figure 2: Test of Trends

sample no.	sample size	Spearman -statistic	Mann -statistics	test statics 5%-level	result
1	4000	2.06	1.98	1.96	trend
2	2850	4.65	4.53	1.96	trend
3	2849	0.65	0.61	1.96	no trend

The next step in the exploratory analysis is generally to see whether there exists a serial correlation between successive I_j's. If no positive indications are obtained, then one can assume that the I_j's are identical distributed, with unknown distribution function. The analysis can only be applied to stationary series, so we first analyse the series without trends and then the remainder must be detrended (see below: modelling of the arrival process). The 22 samples are examined with Wald's non-parametric product-moment-statistics /SIEGEL 56/, and all time series are autocorrelated. See e.g. figure 3.

Figure 3: Test of Autocorrelation

sample no.	sample size	non-parametric statistics	test statics 5%-level	result
1	4000	9,20	1,96	autocorrelated
2	2850	11,13	1,96	autocorrelated
3	2849	6,40	1,96	autocorrelated

3. Arrival Process Modelling

The result of the preceding analysis is a non-stationary autocorrelated stochastic process. The reason for these properties can be explained by the unsteady filling of the system at different daytimes.

An adequate model for the process discovered is a non-homogeneous Poisson-process NPP. The NPP is a generalization of the homogeneous Poisson process HPP. The main difference is the time dependent rate function $\lambda(t)$ of the NPP. Different books and papers discuss NPP's, for example /CINLAR 75/, /COX 66/. The main advantages of the NPP are:

- quality of models can be tested with efficient methods (see below)
- comparison between HPP-modelling and NPP-modelling is simply performed, as a result, the consequence of using simplified models, like HPP, can be evaluated
- random generation of arrivals in simulation experiments or object system experiments are simply executed.

Generally the functional relationship $\lambda(t)$ is unknown and an approximative function is assumed. Estimation of parameters in these models is done in an ad hoc manner.

Another but equivalent description of the arrival process
$(I_j)_{j\in J}$, $I\in\mathbb{R}_+$, $J\subseteq\mathbb{N}_0$, is the point process notation
$(N_t)_{t\in J}$, $N\in\mathbb{N}_0$, $J\subseteq\mathbb{R}_+$. N_t is the number of arrivals until t.

The expected value of N_t is

$$E(N_t) = a(t) = \int_0^t \lambda(\nu)d\nu.$$

Under the assumption that a(t) is a continuous non-decreasing function, it can be shown (see /CINLAR 75/), that the NPP can be transformed into a HPP with $\lambda = \lambda(t) = 1$. This property will be used for testing the quality of parameter estimation and for generating arrival times with pseudo-random mechanisms.

Several models for the time dependent rate function for the NPP have been suggested, but due to mathematical intractabilities only special cases with a few parameters have been treated. We will follow Cox and Lewis /COX 66/ in the use of exponential polynomials

$$\lambda(t) = \lambda(\vec{\alpha},t) = \exp\left(\sum_{j=0}^{m} \alpha_j t^j\right), \tag{2.1}$$

since these functions can approximate any continuous rate arbitrarily closely and always have rate values greater or equal to zero.

The density of arrival times with equation (2.1) is now:

$$f_T(t_i|t_{i-1},\ldots,t_1) = \exp\left(\sum_{j=0}^{m} \alpha_j t_i^j\right)\cdot\exp\left(-\int_{t_{i-1}}^{t_i} \exp\left(\sum_{j=0}^{m} \alpha_j \nu^j\right)d\nu\right). \tag{2.2}$$

The maximum likelihood principle will be used for estimating the parameters, α_j, j=o,...,m, since this method provides asymptotically consistent estimations.

Denoting $t_1,t_2,\ldots,t_k$ arrival times, the likelihood function for $(0,t_k)$ is

$$L = \prod_{i=1}^{k} \lambda(t_i)\exp\left(-\int_0^{t_1} \lambda(\nu)d\nu\right)\ldots\exp\left(-\int_{t_{k-1}}^{t_k} \lambda(\nu)d\nu\right)$$

$$= \prod_{i=1}^{k} \lambda(t_i)\exp\left(-\int_0^{t_k} \lambda(\nu)d\nu\right). \tag{2.3}$$

The natural logarithm of (2.3) with equation (2.1) is:

$$\ln L = \sum_{j=0}^{m} \alpha_j y_j - \int_0^{t_k} \exp\left(\sum_{l=0}^{m} \alpha_l \nu^l\right)d\nu$$

with

$$y_j = t_1^j + t_2^j + \ldots + t_k^j \;,\; j = 0,\ldots,m. \tag{2.4}$$

Differentiation of equation (2.4) yields:

$$\frac{\partial \ln L}{\partial \alpha_j} = y_j - \int_0^{t_k} \nu^j \exp\left(\sum_{l=0}^{m} \alpha_l \nu^l\right)d\nu = 0,\; j = 0,\ldots,m. \tag{2.5}$$

The solution vector $\hat{\vec{\alpha}} = (\hat{\alpha}_0,\ldots,\hat{\alpha}_m)$ is the maximum likelihood estimator of $\vec{\alpha}$.

An appropriate degree m of the polynomial can be obtained by comparing the least square values of the empirical and approximative rate functions:

$$m\varepsilon\{m^*|\min_{1}\left(\sum_{i=1}^{k} [\lambda(t_i)-\lambda(\vec{\alpha},t_i,1)]^2\right) = \sum_{i=1}^{k} [\lambda(t_i)-\lambda(\hat{\vec{\alpha}},t_i,m^*)]^2; m^*\varepsilon\{1,2,\ldots,k-1\}\}.$$

The above procedures were implemented on a DEC-System 1050 in ALGOL 60 and FORTRAN. They work correctly for polynomials up to degree 12 (at least, higher degree have not been tested).

Model Quality

The quality of the approximation can be shown for the transformed HPP. The distribution of the transformed interarrival times is a negativ exponential function with rate $\lambda = 1$. It is proven with Kolmogorov-Smirnov's goodness-of-fit test.

Different time series can sufficiently be approximated. Time series which don't show the statistical properties needed can be approximated piece by piece with the same exponential rate function.

Relation (2.5) is a nonlinear system of equations and cannot be solved explicitly. Approximation techniques, like nonlinear optimization methods without restrictions, must be used.

An adequate minimization function can be obtained:

$$g(\vec{\alpha},t) := \sum_{j=0}^{m} (y_j - \int_0^{t_k} \nu^j \exp(\sum_{l=0}^{m} \alpha_l \nu^l) d\nu)^2 = 0. \qquad (2.6)$$

Since gradient methods generally yield better convergence, we evaluate the derivation of equation (2.6):

$$\frac{\partial g(\vec{\alpha},t)}{\partial \alpha_r} = \sum_{j=0}^{m} 2(y_j - \int_0^{t_k} \nu^j \exp(\sum_{l=0}^{m} \alpha_l \nu^l) d\nu) \cdot (- \int_0^{t_k} \nu^{j+r} \exp(\sum_{l=0}^{m} \alpha_l \nu^l) d\nu) \qquad (2.7)$$

$r = 0,\ldots,m.$

In order to get the minimum of the objective function $g(\vec{\alpha},t)$, with unknown parameter vector $\vec{\alpha}$, we need a starting vector close to the solution of the optimization, i.e. in the convex neighbourhood to ensure the convergence to the local minimum.

McLean /McLEAN 74/ proposed a two step method for determining a starting vector $\vec{\alpha}^0$. His procedure consists of finding an ordinary polynomial representation of the same degree as $\lambda(t)$ which has the observed sums of powers $\{y_j\}$ for its moments.

The polynomial constitutes a pseudo-rate function which can be approximated by an exponential polynomial, after taking logarithms, again by fitting moments. The procedure proposed yields a good starting point for the following minimization.

Different unconstrained optimization techniques are known; however, Himmelblau /HIMMELBLAU 72/ shows that a stringent relationship between the type of the objective function and the optimization method exists. In this context, the best convergence can be obtained by the Davidon-Fletcher-Powell method compared to "projected-Newton-Raphson" or "Marquardt's procedures" (see /HIMMELBLAU 72/). The unidimensional search within this algorithm is done by Fibonacci's method.

Figures4,5,6 show different approximation steps of a time series with sample size k = 4000. Figure 4 with an exponential polynomial with degree 1 (λ= const), figure 5 with 12 degrees and figure 6 with 6 intervals and a maximum of 6 degrees. The last approximation is close to the rate function measured but put out of sight the typical trends of day. In figure 7 some values of the statistical analysis for the transformed process are demonstrated:

Figure 4: Ratefunction
exponential polynomial approximation, 1 degree

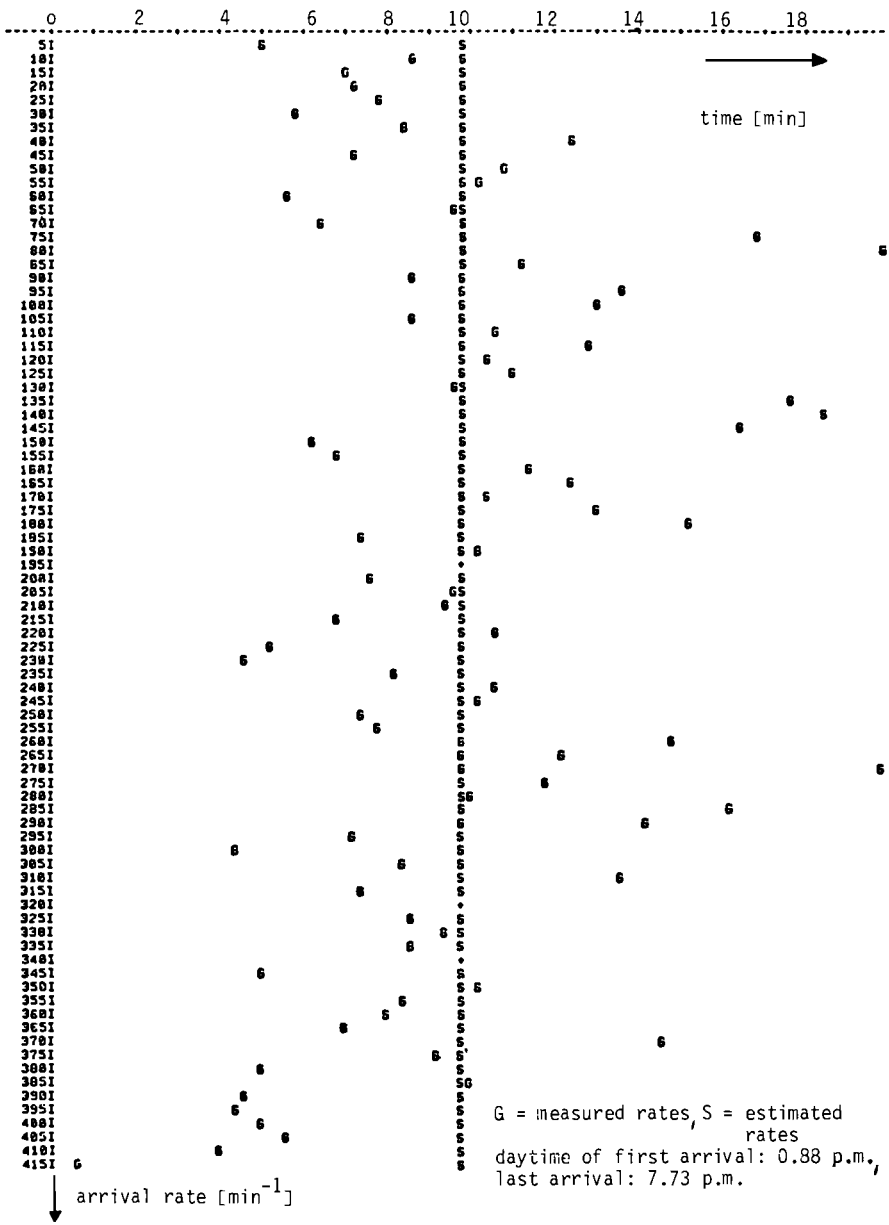


Figure 5: Ratefunction
exponential polynomial approximation, 12 degrees
G = measured rates, S = estimated rates, + $\hat{=}$ G = S

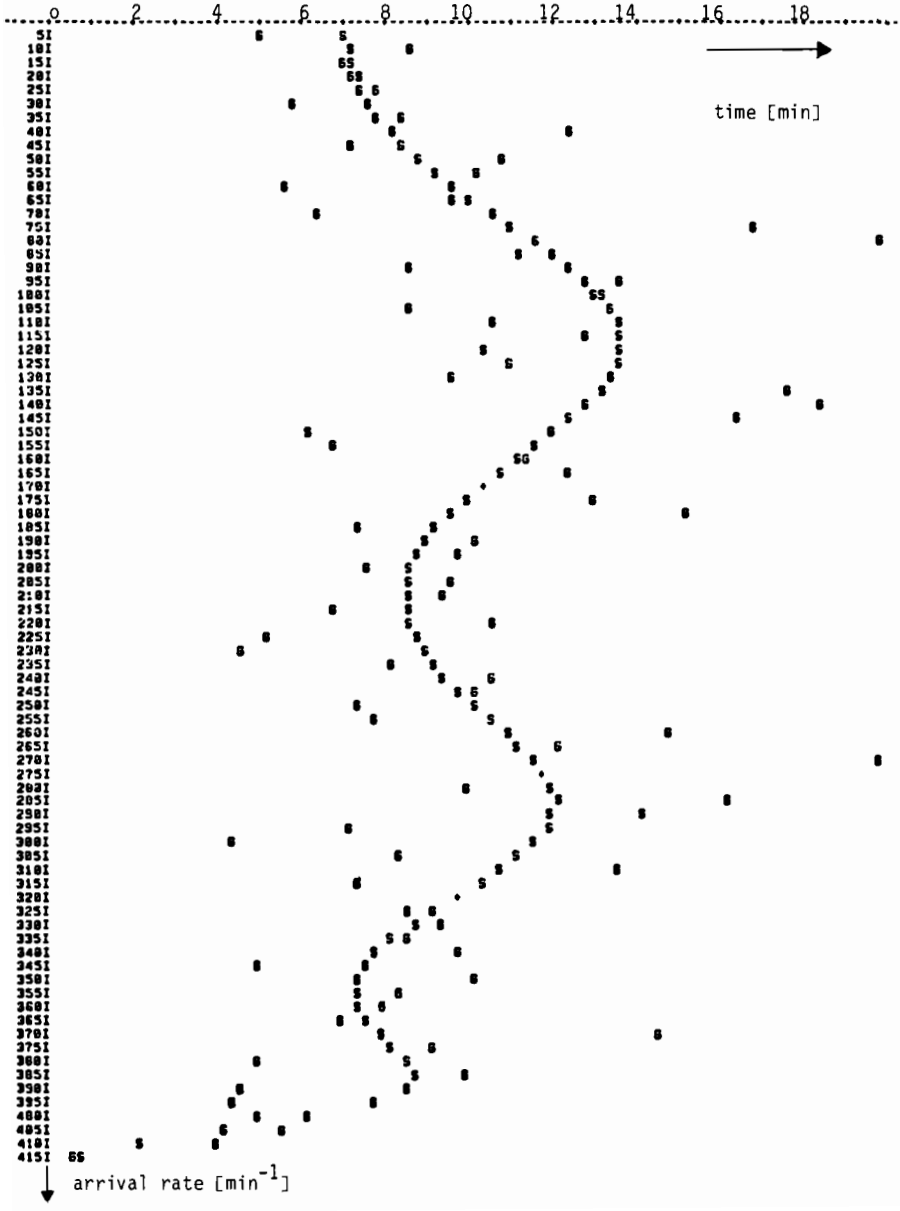


Figure 6: Ratefunction
exponential polynomial approximation, 6 intervals, max. 6 degrees
G = measured rates, S = estimated rates, + ≙ G = S

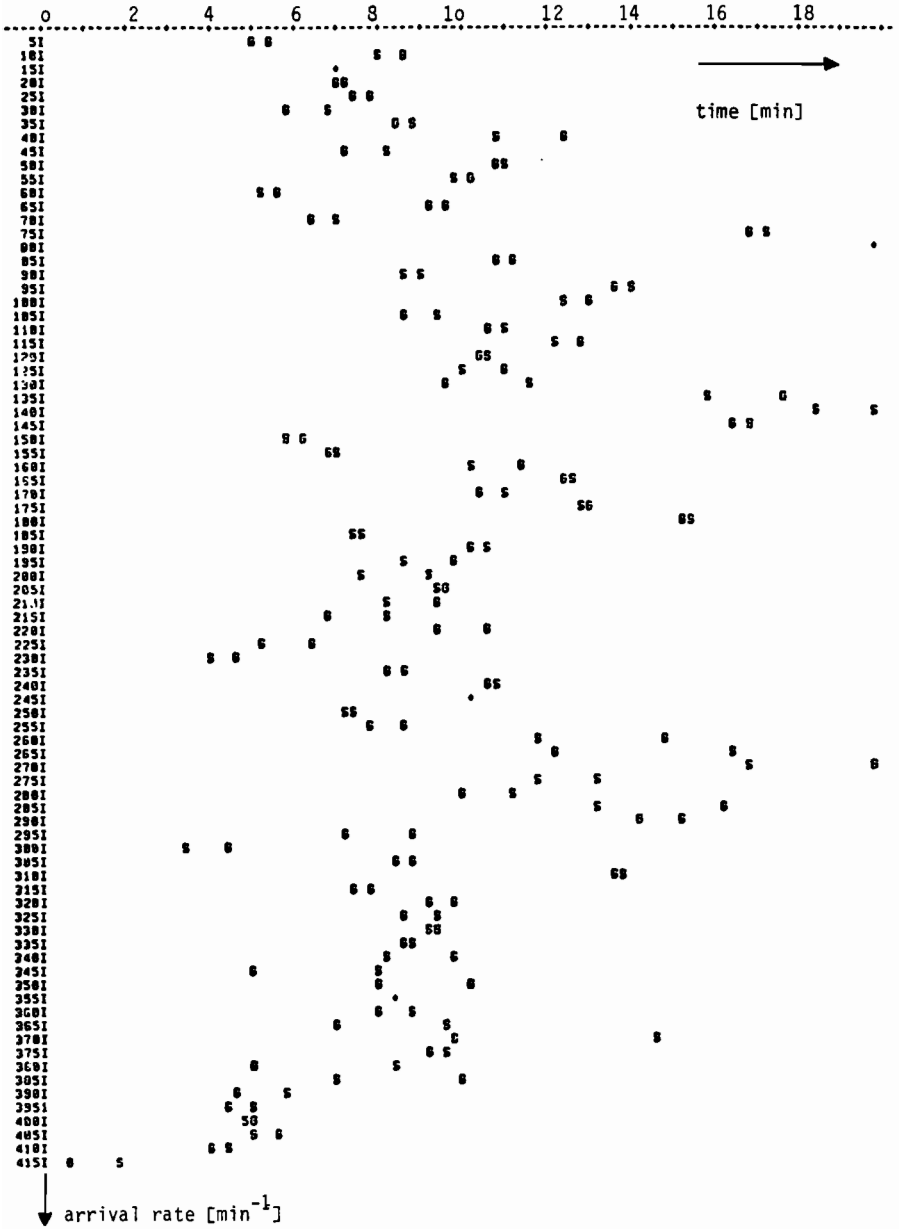


Figure 7: Arrival Process Analysis (k = 4000)

	approximation with 12 degrees	interval approximation
$1/\lambda$	1.004	1.005
test of trend	no trend	no trend
test of autocorrelation	autocorrelation	no autocorrelation

The methods proposed in the preceding parts are suitable techniques to construct valid arrival process models for predefined intervals. The overall execution time for a time series with sample size k=4000 is about 1,5 hours (executed at a DEC System 1050), one hour for the statistical analysis (the time consuming part is the evaluation of the autocorrelation coefficients up to lag 50) and about 30 minutes evaluating a starting vector and the following minimization.

3. Work Demand Analysis and Modelling

In addition to the description of the arrival process a statistical analysis and modelling of job demands $\{\vec{X}_j\}_{j\in J}$, $J\subseteq \mathbb{N}$, $\vec{X}_j\in \mathbb{R}_+^p$, is needed.

The stochastic independence of jobs is assumed according to the low relationships between different job demands (k = number of jobs):

$$f(\vec{x}_1,\vec{x}_2,\ldots,\vec{x}_k) = f(\vec{x}_1)\cdot f(\vec{x}_2)\cdot\ldots\cdot f(\vec{x}_k).$$

Analogously to the preceding part we start with a statistical analysis of the work demands:

- descriptive statistical analysis,
- independence and correlation analysis.

In this work the following characteristics were collected for each job:

X_1: execution time [s]

X_2: core requirements [kwords]

X_3: magnitude of disk input [blocks] (1block=128words)

X_4: magnitude of disk output [blocks]

X_5: magnitude of TTY input [characters]

X_6: magnitude of TTY output [characters]

The graphs of the unidimensional density functions of the random variables $X_1,\ldots X_6$ indicate multimodal properties for a few variables (X_2,X_3,X_5,X_6).

The demand description can be simplified if the job variables $X_1,\ldots X_p$ are mutually independent, i.e.

$$f(x_1,\ldots,x_p) = f(x_1)\cdot f(x_2)\cdot\ldots\cdot f(x_p).$$

In order to test the independence hypothesis we apply a generalization of the usual two-dimensional contingency table method /VICTOR 70/. The analyis of all samples results in a dependence between all variables $X_1,\ldots X_6$. Since we are interested in the degree of association a correlation analysis follows by using the multiple correlation coefficient /MORRISON 67/. The different samples indicate strong relationshps between all variables (see example in figure 8).

As a result of the statistical analysis of job demands a multivariate distribution function is needed which allows different modes.

Figure 8: Correlation Analysis (sample size 4000)

dependent variable	independent variables	R...	f	$F_{0.005,5,3994}$
X_1	X_2,X_3,X_4,X_5,X_6	0.656	603.5	
X_2	X_1,X_3,X_4,X_5,X_6	0.240	48.9	
X_3	X_1,X_2,X_4,X_5,X_6	0.323	92.9	1,15
X_4	X_1,X_2,X_3,X_5,X_6	0.269	62.5	
X_5	X_1,X_2,X_3,X_4,X_6	0.128	13.3	
X_6	X_1,X_2,X_3,X_4,X_5	0.576	398.5	

Work Demand Modelling

In order to take into account the properties of the work demands, as demonstrated in the foregoing section, a mixture of density functions is proposed. A distribution or density mixture arises when a population is made up of component populations mixed together in fixed proportions. The problem is to decompose the mixture by estimating any unknown parameters of the component densities, the mixing proportions, and the number of components (under the assumption of a known density family of the components), given a sample of observations from the mixed populations.

Let m be the number of components, let $f_i(\vec{x},\vec{\lambda}_i)$ denote the density function of the i-th component of random vector $\vec{X}$ with parameter vector $\vec{\lambda}_i$, and let $p_i(\vec{\lambda}_i)>0$ be the proportion of the i-th component in the mixture. The mixed density function can be written as

$$f_{\vec{X}}(\vec{x}) \equiv f(\vec{x}) := \sum_{i=1}^{m} p_i(\vec{\lambda}_i)\cdot f_i(\vec{x},\vec{\lambda}_i), \quad \sum_{i=1}^{m} p_i(\vec{\lambda}_i) = 1.$$

The estimation of the mixture may be interpreted as a problem of "unsupervised estimation" or "learning without a teacher". These kinds of problems are frequently found in the area of pattern recognition, thus, an extensive literature can be found there (compare /DUDA 13/). However, there is still no reliable, general-purpose estimation procedure available. The difficulty of the estimation problem depends on the number of components, the extent to which they overlap, and the particular combination of parameters to be estimated. Only in simplified cases under additional assumptions (e.g. known number of components or proportions) satisfactory algorithms are known.

The proposed two step estimation method attempts to overcome most of the listed difficulties. It works satisfactorily for the underlying modelling of job demands. - It is assumed that the component distributions are multivariate and normal functions with unknown parameters. Firstly, a mixture of multivariate normal distributions is identifiable, i.e. there is only one solution to fit the data /YAKOWITZ 70/. Secondly, a simple pseudo-random mechanism to generate random vectors with dependent variables is available. Under the assumption of identifiability, other component distributions can be used.

The first step of the algorithm guides in selecting an initial estimation of the number of components and the remaining parameters. In the second step the initial estimation is improved using a maximum likelihood criterion. Since the costs of the numerical calculations are very high for large samples a particular form fo the mixture model is applied. It can be proven that a conditional mixture model working with memory significantly reduces the cost /MATERNA 78/. Thus,

each demand vector $\vec{X}$ is extended by a membership vector $\vec{Z}$, which denotes the relationship between the vector $\vec{X}$ and its underlying component.

Initial State Estimation

Starting the algorithm we are looking for points $\vec{X}_j$ with local object concentration in the multidimensional space. Assuming normally distributed components the density can be written:

$$f(\vec{x}) = \sum_{i=1}^{m} p_i \cdot f_i(\vec{x},\vec{\mu}_i,\overleftrightarrow{\Sigma}_i),$$

$\vec{\mu}_i$ = mean vector of component i,

$\overleftrightarrow{\Sigma}_i$ = covariance matrix of component i.

In order to estimate the number of components, we shall use Schnell's idea /SCHNELL 64/ and approximate the mixture density by a superposition of k p-dimensional normally distributed densities.

$$\hat{f}(\vec{x}_1) := \sum_{j=1}^{k} \frac{1}{(2\Pi)^{p/2}\sigma^p} \exp\left[\frac{||\vec{x}_1-\vec{x}_j||^2}{2\sigma^2}\right], \quad l\varepsilon\{1,\ldots,k\},$$

with the distance $||\vec{x}_1-\vec{x}_j||^2 = \sum_{i=1}^{p} (x_{1i}-x_{ji})^2$.

The function $\hat{f}(\vec{x}_1)$ is maximized by shifting each point $\vec{x}_1$ in the direction of steepest ascend, i.e. each point is removed to its nearest mode.

The moving

$$\vec{x}_1^{\,(n+1)} = \vec{x}_1^{\,(n)} + \beta\vec{s}$$

is continued until $f(\vec{x}_1^{\,(n+1)})-f(\vec{x}_1^{\,(n)}) \leq 0$ failes,

with $\vec{s}$ as the direction of steepest ascend:

$$\vec{s} := \nabla f(\vec{x}_1^{\,(n)}) \Big/ ||\nabla f(\vec{x}_1^{\,(n)})||, \quad ||\nabla f|| > 0,$$

and the step size β. The value $\beta \approx \sigma/2$ is an empirically determined reasonable step size. As a consequence the whole sample S can be decomposed into mutually exclusive sample sets S_i : $S = \bigcup_{i=1}^{m} S_i$. This is done using the euclidian distance measure:

$$d_{lh} = ||\vec{x}_1 - \vec{x}_h||,$$

since all points were removed into the 2β surroundings of the different modes. Each object is associated with one set S_i, thus, we can evaluate the parameters $p_i,\vec{\mu}_i,\overleftrightarrow{\Sigma}_i$, i=1,...,m.

Final Parameter Estimation

The maximum likelihood principle is used to improve the initial estimates because asymptotically consistent estimations can be obtained. The likelihood functions of an object $\vec{X}\varepsilon S$ is

$$L(P,\vec{x}) = f(\vec{x},P) = f(\vec{x}), \quad P = \{\vec{\mu}_1,\overleftrightarrow{\Sigma}_1,\ldots,\vec{\mu}_m,\overleftrightarrow{\Sigma}_m,p_1,\ldots,p_m\}.$$

The following function is obtained for a sample $S = S^k$ with size k:

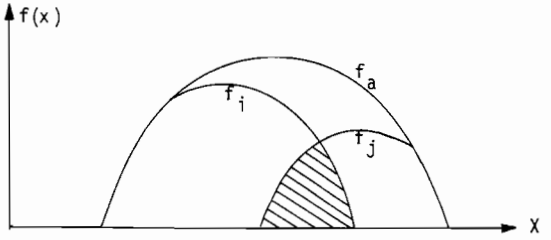
$$L(P,S^k) = \prod_{j=1}^{k} f(\vec{x}_j,P) = \prod_{j=1}^{k} \sum_{i=1}^{m} p_i \cdot f_i(\vec{x}_j,\vec{\mu}_i,\overleftrightarrow{\Sigma}_i). \qquad (3.1)$$

The usual procedure (differentation of equation (3.1)) used in maximum likelihood estimation is intractable. Therefore, a step by step maximization is proposed. The association between $\vec{x}_i$ and S_i is known and after taking the logarithm, equation (3.1) can written as:

$$\begin{aligned} \ln L(S) &= \sum_{j=1}^{k} \ln \sum_{i=1}^{m} p_i f_i(\vec{x}_j,\vec{\mu}_i,\overleftrightarrow{\Sigma}_i) \\ &= \sum_{j=1}^{k} \ln \sum_{i=1}^{m} q_{ji}(\vec{x}_j,p_i,\vec{\mu}_i,\overleftrightarrow{\Sigma}_i), \text{ whereby } q_{ji}=p_i f_i(\vec{x}_j) \\ &= \sum_{r=1}^{m} \sum_{\vec{x}_j \in S_r} \ln \sum_{i=1}^{m} q_{ji} \\ &= \sum_{r=1}^{m} l_r, \text{ whereby } l_r = \sum_{\vec{x}_j \in S_r} \ln \sum_{i=1}^{m} q_{ji}. \end{aligned}$$

The value of l_i is the likelihood of the component i. The l_i's are used to agglomerate non-significant components. An agglomeration $S_a = S_i \cup S_j$ of two components is performed if the likelihood $\ln L(P,S^k)$ can be maximized (compare figure 9).

Figure 9: Agglomeration of two components i,j



A numerical simplification is obtained if we use the condition $l_a \geq l_i + l_j$ instead of equation (3.1)

Furthermore, we will reorganize single objects to maximize the likelihood. This is supported on the following membership function:

$$h_i(\vec{x}_j) := \exp\left[-\frac{1}{2}(\vec{x}_j-\vec{\mu}_i)^T \overleftrightarrow{\Sigma}_i^{-1}(\vec{x}_j-\vec{\mu}_i)\right], \; i = 1,\ldots,m, \; 0<h_i\leq 1,$$

which measures the membership of the object $\vec{x}_j$ to a component i. $h_i = 1$ if $\vec{x}_j = \vec{\mu}_i$, and h_i is nearly zero if $\vec{x}_j$ is far away from $\vec{\mu}_i$. Now the membership condition can be written as:

$$\vec{x}_j \in S_i : h_i(\vec{x}_j) = \max\{h_1(\vec{x}_j),\ldots,h_m(\vec{x}_j)\}, \; i = 1,\ldots,m \; j = 1,\ldots,k.$$

In order to increase the likelihood, the agglomeration and reorganization are repeated iteratively. - As an example one hour execution time of a DEC-System 1050

is needed to approximate a sample with size k = 1000 and 6 dimensions.

Figure 10a,b display some results for an artificial sample with known parameters. Another example is provided in figure 11. It is a two-dimensional vector of job demands based on.

(i) generation under the assumption of independent demands,
(ii) generation with a multivariate normal distribution,
(iii) generation with a mixed distribution.

In order to demonstrate the results more clearly we use a discrete distribution with 25 two-dimensional intervals and compute the sum of the squared differences A between measured and generated objects for each cell.

4. Conclusion

Different statistical methods are proposed which allow a valid description of the real workload based on the modelling of the job arrival process and a multivariate mixed distribution model of job demands. In both cases a random generation of arrivals and demands is possible and can be applied for simulation experiments and for calibrating synthetic jobs. Since we obtain a valid description of the workload the cost of the numerical calculations are acceptable.

Acknowledgements

The author is indebted to Prof. E. Jessen and Prof. H. Beilner for many fruitful discussions and their encouragement in performing this study.

Figure 10a: Approximation of a two-dimensional artificial sample with three components

real values:			estimated values:		
$\vec{\mu}_1 = (10 \quad 10)^T$	$k_1 = 50$		$\vec{m}_1 = (10.157 \quad 10.075)^T$	$\hat{k}_1 = 51$	
$\overleftrightarrow{\Sigma}_1 = \begin{pmatrix} 4 & 1.732 \\ 1.732 & 1 \end{pmatrix}$			$\overleftrightarrow{C}_1 = \begin{pmatrix} 3.929 & 1.919 \\ 1.919 & 1.162 \end{pmatrix}$		
$\vec{\mu}_2 = (10 \quad 13)^T$	$k_2 = 50$		$\vec{m}_2 = (9.910 \quad 12.948)^T$	$\hat{k}_2 = 49$	
$\overleftrightarrow{\Sigma}_2 = \begin{pmatrix} 0.25 & 0 \\ 0 & 0.25 \end{pmatrix}$			$\overleftrightarrow{C}_2 = \begin{pmatrix} 0.233 & -0.015 \\ -0.015 & 0.216 \end{pmatrix}$		
$\vec{\mu}_3 = (14 \quad 13)^T$	$k_3 = 50$		$\vec{m}_3 = (13.961 \quad 13.055)^T$	$\hat{k}_3 = 50$	
$\overleftrightarrow{\Sigma}_3 = \begin{pmatrix} 4 & -1.732 \\ -1.732 & 1 \end{pmatrix}$			$\overleftrightarrow{C}_3 = \begin{pmatrix} 3.960 & -1.528 \\ -1.528 & 0.789 \end{pmatrix}$.		

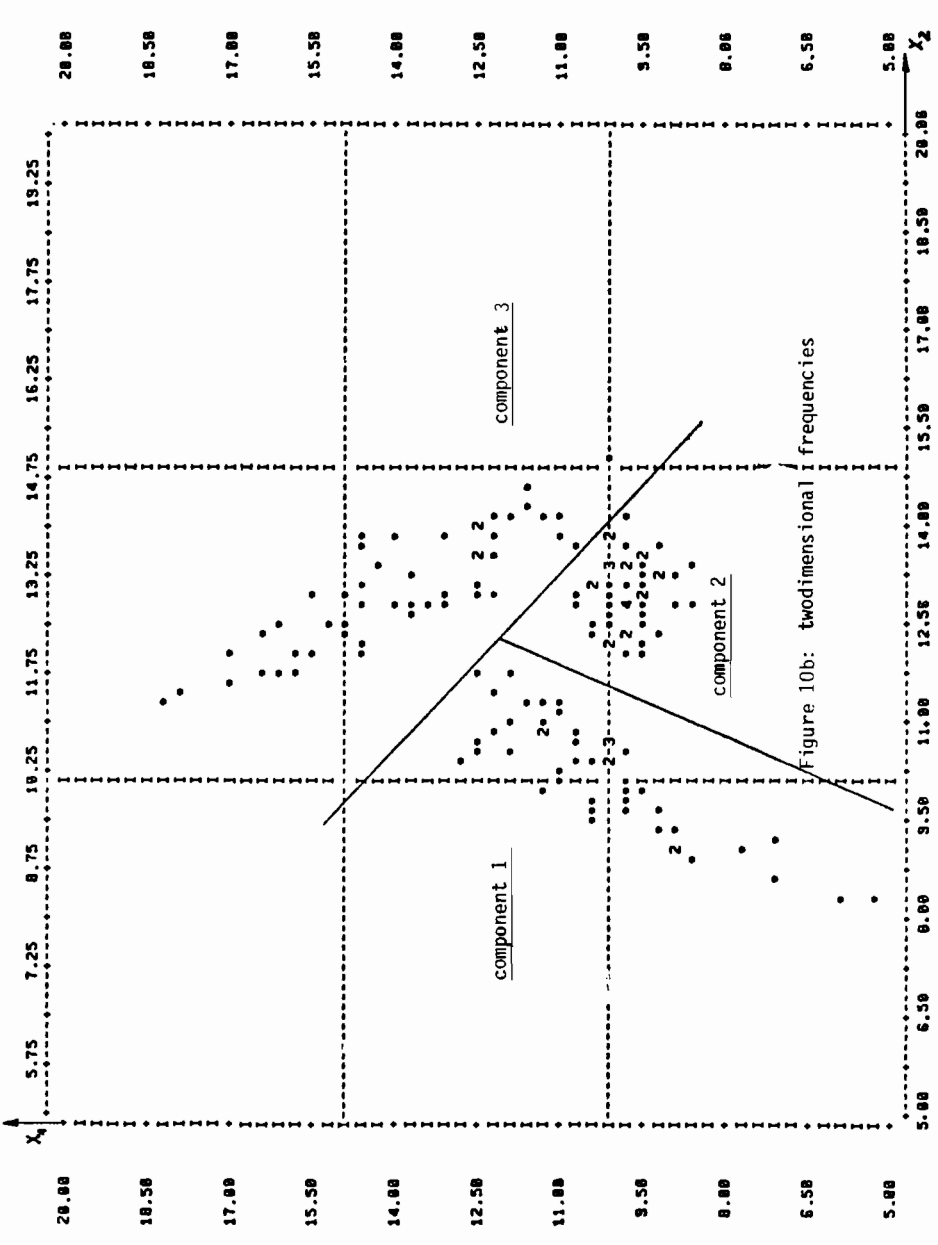


Figure 10b: twodimensional frequencies

Figure 11: Comparison of measured and generated demands
(X_1 = CPU-time, X_2 = core requirements)

			measured demands	generated demands independence assumption	multivariate normal distr.	mixed distribution
1	1	1	126	31	291	143
2	1	2	371	227	265	368
3	1	3	204	149	211	226
4	1	4	3	23	120	7
5	1	5	11	63	49	14
6	2	1	11	36	14	29
7	2	2	311	327	21	346
8	2	3	343	260	23	341
9	2	4	12	60	15	14
10	2	5	14	97	9	4
11	3	1	1	17	26	3
12	3	2	271	231	33	241
13	3	3	91	198	40	76
14	3	4	83	28	30	91
15	3	5	114	69	11	103
16	4	1	2	16	68	0
17	4	2	50	166	110	44
18	4	3	89	126	120	49
19	4	4	43	23	85	39
20	4	5	51	47	63	42
21	5	1	4	10	80	9
22	5	2	54	93	123	52
23	5	3	34	76	211	58
24	5	4	43	12	218	33
25	5	5	77	28	177	81
sum of quares A =			—	93 328	406 789	6 236

5. Literature

/AGRAWALA 76/ A.K. Agrawala, J.M. Mohr, R.M. Bryant: An Approach to the Workload Characterization Problem
Computer, June 1976

/ARTIS 78/ H.P. Artis: Capacity Planning for MVS Computer Systems, Performance of Computer Installations
D. Ferrari (ed.), North-Holland, Publishing Company, 1978

/CINLAR 75/ E. Cinlar: Introduction to Stochastic Processes
Prentice-Hall, Inc. 1975

/COX 66/ D.R. Cox, P.A.W. Lewis: The Statistical Analysis of Series of Events
Wiley a. Sons, New York, 1966

/DUDA 73/ R.O. Duda, P.E. Hart: Pattern Classification and Science Analysis
J. Wiley a. Sons, 1973

/FANGMEYER 76/ H. Fangmeyer, R. Gloden, J. Larisse: An Automatic Clustering Technique Applied to Workload Analysis and System Tuning,

Proceedings: Modelling and Performance Evaluation of Computer Systems
Ispra (Italy), 1976, North-Holland Publishing Company, 1977

/HIMMELBLAU 72/ D.M. Himmelblau: Applied Nonlinear Programming
McGraw-Hill, 1972

/HOLLANDER 73/ M. Hollander, D.A. Wolfe: Nonparametric Statistical Methods
Wiley a. Sons, 1973

/LANDAU 76/ K. Landau: Clusteranalytische Untersuchungen zur Computer-Arbeitslast
Elektr. Rechenanlagen, 18, 1976

/MATERNA 78/ W. Materna: Ein allgemeines Verfahren zur Konstruktion von repräsentativen Lastmodellen, Dissertation
Bericht Nr. 43, Fachbereich Informatik, Universität Hamburg

/MACLEAN 74/ C.J. MacLean: Estimation and testing of an exponential polynomial rate function within the nonstationary Poisson Process, Biometrica, 61 1974

/MORRISON 67/ D.F. Morrison: Multivariate Statistical Methods
McGraw-Hill, 1967

/SACHS 73/ L. Sachs: Angewandte Statistik
Springer-Verlag, Berlin, 1974

/SCHNELL 64/ P. Schnell: Eine Methode zur Anfertigung von Gruppen
Biom. Zeitschrift, 1964

/SIEGEL 56/ S. Siegel: Nonparametric Statistics for the Behaviorial Sciences
McGraw-Hill, New York, 1956

/SREENIVASAN 74/ K. Sreenivasan, A.J. Kleimann: On the Construction of a Representative Synthetic Workload
CACM, March 1974

/VICTOR 70/ N. Victor: Analyse mehrdimensionaler Kontingenztafeln
Dissertation 1970, Universität Mainz

/YAKOWITZ 70/ S.J. Yakowitz: Unsupervised Learning and the Identification of Finite Mixtures
IEEE Trans. IT, May 1970

SCHEDULING TECHNIQUES

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

SCHEDULING UNDER RESOURCE CONSTRAINTS - - ACHIEVEMENTS AND PROSPECTS

J.Błażewicz , J.Węglarz
Institute of Control Engineering
Technical University of Poznań
Poznań, Poland

1. INTRODUCTION

In designing computer operating systems an important factor is the choice of the appropriate scheduling algorithm used to deter - mine the order in which tasks are to be executed. One approach to the analysis of scheduling algorithms consists in developing an abstract model of a computing system and then in the formal analy - sis of the algorithms operating in the model framework. We will be concerned with a deterministic model of a computing system, i.e. one in which no random variable appears. In particular, task execu- tion times are known a priori. An interpretation of this assumption as well as results obtained under it are given, for example, in [11]. In most cases,also the number of tasks and task ready times are assumed to be known a priori, however, even if this is not the case, they are not described in a probabilistic way.

Until now, most of the results obtained in deterministic compu- ter scheduling theory have concerned scheduling on processore with- out additional resources, and there exist several papers and books which survey these results [11,22,34,39]. In this paper we would like to survey and extend results concerning the problems of sche- duling tasks on processors under resource constraints which have been intensively studied recently. Our attention will be devoted mainly to the model in which the number of tasks, task ready and execution times and resource requirements are known a priori. More- over, we will assume that tasks are nonpreemptable, and in the case when preemptions are allowed - that all resources are preemptible, i.e. they are released at the moment of the preemption. The last assumption implies that deadlock cannot occur. Of course, it may easily be noticed that the above assumptions are rather seldom fulfilled strictly in real situations. The analysis of such a model has, however, an important theoretical and practical significance. Generally speaking, from the theoretical point of view it allows for the marking out of the "border" between "easy" and "hard", i.e. NP-complete problems. On the other hand, in practice optimal stra- tegies obtained under strong assumptions suggest philosophies for heuristic procedures under weaker assumptions.

Mention will also be made of other models, the complete discus- sion of which would require separate surveys, which could in any case be very difficult to make at present. Consequently, our com - ments about these models should be treated as indicating directions for further research.

The basic model of a computing system which we are concernsd

with will be described in Section 2. Then, we preeent a complexity enalyeie of scheduling problems for this model. This involvee either giving a polynomial-in-time optimization algorithm which alwaye finds an optimal echedule or proving this problem to be NP-complete, thus unlikely to admit of such an algorithm (we refer the reader to [1,21,43] for an introduction to the theory of NP-completeneee). Such an analyeis is carried out in Section 3, for three main performance measuree: schedule length, mean flow time and maximum lateness. It is obvious that in practice mainly non-enumerative algorithms can be directly used in operating systems. Thue, complexity analysis justifies the use of polynomial-in-time approximation algorithms whoea woret caee behavior is known, for solving NP-complete probleme. Examples of such algorithms are preeented in Section 4. Laetly, in Section 5 we preeent enumerative optimization methods which may be useful in evaluating approximation algorithms or in syeteme which allow for the utilization of off-line computational reeults.

Other modele of computing systems are considered in Section 6. Theee models include : echeduling in systems where each processor has its own primary memory rather than accees to a common one, the simultaneoue liquidation of eystem performance failures and minimization of mean weighted flow time, and echeduling with memory allocation in a certain claes of multiproceesing systeme.

2. BASIC MODEL AND SOME DEFINITIONS .

Our basic model is concerned with a computing eystem consisting of parallel processors and some other reeources such ae: main and mass storage, line printers, card readers and punchers, teletypes, etc. Such a system may be described by three components: a set of n tasks $\{T_1,T_2,\ldots,T_n\}$, which are to be processed on m parallel and identical processore $\{P_1,P_2,\ldots,P_m\}$, utilising additional resources $\{R_1,R_2,\ldots,R_s\}$. For each resource R_l , $l=1,2,\ldots,s$, there exists a bound m_l which gives the total amount of that reource available at any given time. For every task T_j, $j=1,2,\ldots,n$ there are given: execution time p_j, ready time r_j , deadline d_j and a vector $R(T_j)$ containing resource requirements (the l-th element of this vector $R_l(T_j) \leq m_l$, denotes the number of units of R_l required by task T_j). A partial order $\prec$, specifying precedence constraints is defined on the task set: $T_i \prec T_j$ means that T_j cannot begin processing until T_i is finished.

Some other definitions will be useful in the following discusion. Tasks are called independent if there is no partial order imposed on the task set, otherwise they are called dependent. By preemptable we mean tasks that may be preempted at any moment and restarted later (maybe on another processor) with no time losses. Tasks that cannot be preempted are called nonpreemptable. A schedule is e specification of the assignment of processors and additional resources to the tasks, and this assignment must satisfy the following conditions:

- all tasks are processed to completion that is they are assigned all required resources during the time they are being processed, and thees resources are released either at the end of the processing or at the moment of the preemption,
- for each $T_i \prec T_j$, T_j is begun after T_i is completed,

- at most m tasks are processed at a time,
- for each time t, $\sum_{T_j \in A(t)} R_l(T_j) \leq m_l$, where $A(t)$ is the set of tasks which are being processed at time t.

For every task T_j in the schedule we denote its completion time by C_j. To evaluate schedules we will use three performance measures: schedule length $C_{max} = \max_j \{C_j\}$, mean flow time $\bar{F} = \sum_{j=1}^{n} C_j/n$, and maximum lateness $L_{max} = \max_j \{C_j - d_j\}$. The schedule will be called optimal if the value of the respective measure is minimal for it. A scheduling algorithm is a step by step procedure that produces a schedule for every given set of tasks. By optimization scheduling algorithm we mean an algorithm which always finds an optimal schedule, and by approximation scheduling algorithm - one which, while not always finding optimal schedules, tends to find schedules that are close to optimal. Preemptive and nonpreemptive scheduling algorithms are those for scheduling preemptable and nonpreemptable tasks respectively.

In this paper, we consider several scheduling subproblems which differ from each other by their task, processor and scheduling characteristics. To distinguish these subproblems we use some elements of the notation $\alpha \mid \beta \mid \gamma$ [22,35], where ($\emptyset$ denotes the empty symbol):

$\alpha = \alpha_1 \alpha_2$: describes the processor environment;
- $\alpha_1 = P$: identical parallel processors;
- $\alpha_2 = m$: number of processors is equal to m,
- $\alpha_2 = \emptyset$: number of processors is assumed to be variable.

$\beta = \beta_1, \beta_2, \beta_3, \beta_4, \beta_5$: describes task characteristics;
- $\beta_1 = pmtn$: tasks are preemptable,
- $\beta_1 = \emptyset$: no preemption is allowed ;
- $\beta_2 = res\, \lambda\, \sigma\, \varrho$: characterizes resource requirements;
  - $\lambda, \sigma, \varrho = 1$: denotes respectively: one resource type, unit resource bounds for each resource type, 0 or 1 resource requirements of all the tasks,
  - $\lambda, \sigma, \varrho = \cdot$: denotes respectively: several resource types arbitrary resource bounds, arbitrary resource requirements;
- $\beta_3 = prec$: arbitrary precedence constraints ,
- $\beta_3 = tree$: precedence constraints between the tasks such that the associated precedence graph is a tree,
- $\beta_3 = \emptyset$: no precedence constraints are specified;
- $\beta_4 = r_j$: there are specified ready times that may differ between tasks,
- $\beta_4 = \emptyset$: for all j, $r_j = 0$;
- $\beta_5 = p_j = 1$: each task has unit execution time,
- $\beta_5 = \emptyset$: arbitrary execution times

γ : denotes the optimality criterion, i.e.
- C_{max} : schedule length,

$\sum C_j$: mean flow time,

L_{max} : maximum lateness.

3. COMPLEXITY RESULTS AND SIMPLE OPTIMIZATION ALGORITHMS.

In this Section we present results concerning the complexity of scheduling problems connected with the presented basic model. We will try to establish a sharp berderline betwenn easy problems solvable in polynomial time, and more difficult, NP-complete, ones. Since we are interested in the application of scheduling algorithms in operating systems such a borderline will inform us for what problems we can use simple optimization algorithms and for what problems we should search for simple approximation ones. We start with the schedule length criterion.

For the problem $P|res1 \cdot 1, p_j=1|C_{max}$ one can use the following algorithm.

Algorithm 1

1. Set $t:=0$, $k:=0$.
2. At moment t assign a task for which $R_1(T_i)=1$ to the next free processor and set $k:=k+1$. Repeat this step until $k=m_1$ or there is no such task.
3. Assign to all free processors at moment t tasks for which $R_1(T_i)=0$. Set $t:=t+1$ and $k:=0$, and repeat step 2.

The optimality od the above algorithm is obvious, since it tends at every moment to use the most resources possible. It may be easily implemented in $O(n)$ time.

Let us now turn our attention to the problem $P2|res\cdots, p_j=1|C_{max}$ which can be solved by the use of Algorithm 2 [17].

Algorithm 2

1. Construct an n-node graph G (undirected) , having each node labeled as a distinct task, with an edge joining T_i to T_j if and only if $R_l(T_i)+R_l(T_j) \leq m_l$, $l=1,2,\ldots,s$.
2. Find a maximal cardinality set E of edges from G such that no two edges share a common endpoint and put the minimal value of schedule length $C^*_{max} = n-|E|$.
3. Process in parallel the tasks which are joined by the edges comprising the set E . Process other tasks individually.

Since to obtain set E Kariv and Even's [30] maximal matching algorithm can be used, the complexity of Algorithm 1 is $O(n^{2.5})$. Most of the other problems of scheduling to minimize schedule length are NP-complete as is shown in the following theorem.

Theorem 1

The following problems are NP-complete:

1. $P2|res1\cdot\cdot,tree,p_j=1|C_{max}$ [17]
2. $P2|res111,prec,p_j=1|C_{max}$ [43]
3. $P3|res1\cdot\cdot,p_j=1|C_{max}$ [17]
4. $P|res\cdot11,p_j=1|C_{max}$ [7]
5. $P2|res111|C_{max}$ [36]

It follows that the complexity of some problems remains an open question. This is especially true for the case of preemptable tasks. However, analysing proofs of cases 1,2 and 3 of Theorem 1, one may see that there is no advantage in presmption, so probleme $P2|pmtn,res1\cdot\cdot,tree|C_{max}$, $P2|pmtn,res111,prec|C_{max}$ and $P3|pmtn,ree1\cdot\cdot|C_{max}$ are also NP-complete.

Let us now consider the problem of mean flow time minimization. The problem $P|res1\cdot1,p_j=1|\sum C_j$ can be solved by Algorithm 1, and problem $P2|ree\cdot\cdot\cdot,p_j=1|\sum C_j$ by the following algorithm [3].

Algorithm 3.

1. Find the schedule with the minimal length.
2. Reorder processing in the obtained schedule so that all pairs of tasks are processed first, and after them single tasks.

Since to construct a schedule with minimum length Algorithm 2 can be used, the complexity of the above algorithm is $O(n^{2.5})$. No other optimization algorithm is known. Some of the remaining problems may be proved to be NP-complete.

Theorem 2

The following problems are NP-complete:
1. $P2|res1\cdot\cdot,tree,p_j=1|\sum C_j$ [3],
2. $P2|res111,prec,p_j=1|\sum C_j$ [3]
3. $P3|res1\cdot\cdot,p_j=1|\sum C_j$ [3]
4. $P|res\cdot11,p_j=1|\sum C_j$ [7]

We now turn our attention to the problems of minimizing maximum lateness, which are of value for computer control eystems. Firstly, we show how the problem $P|res1\cdot1,r_j,p_j=1|L_{max}$ can be solved. More clearly, we describe algorithms for determining, for the same input data, whether or not tasks can be scheduled before their deadlines (this problem will be denoted by $P|res1\cdot1, r_j,p_j=1,d_j|-$). This can be done since first problem may be polynomially transformed to the second one. Namely, given an instance of the problem $P|res1\cdot1,r_j,p_j=1|L_{max}$ with deadlines d_j', $j=1,2,\ldots,n$ and a threshold value L for which we ask the question of the existence of a schedule with $L_{max} \leq L$ [1)], we can construct

1) It may be proved [1], that in analysing the complexity of an optimization problem, we can represent it by a decision problem for which an answer may be "yes" or "no".

an instance of $P|res1\cdot1, r_j, p_j=1, d_j|-$ by taking all the parameters the same as in the first problem, except the deadlines for which we have $d_j := d_j' + L$, $j=1,2,\ldots,n$. Obviously, the answer is "yes" for the first problem, if and only if the same answer is for the second one. Thus, it is sufficient to solve the second problem in polynomial time and the first one may be solved as well. To solve the problem $P|res1\cdot1, r_j, p_j=1, d_j|-$ it is necessary to calculate first modified deadlines $\{d_j^*\}$ by the use of Algorithm 4 [6]. Then, these modified deadlines are used in Algorithm 5 [6] which constructs a schedule with no task late, whenever such a schedule exists. Further, we call task T_j active (at moment t) if $r_j \leq t$, and we define set A_t to be $A_t = \{T_j : (d_j^* = t) \wedge (R_1(T_j) = 1)\}$. The two algorithms may be described as follows [6].

Algorithm 4

1. Order tasks in the nondecreasing order of their original deadlines. Initially assign $d_j^* := d_j$ for all tasks. Set $t := \max_j \{d_j\}$.
2. If $|A_t| \leq m_1$ then go to step 3. From the set A_t take $|A_t| - m_1$ tasks with the earliest ready times and calculate their modified deadlines according to the formula:
$$d_j^* := d_j^* - 1.$$
3. Set $t := t-1$. If $|A_t| = 0$ repeat this step until either $|A_t| > 0$ or $t=0$. In the former case go to step 2, in the latter end the algorithm (if $|A_0| \neq 0$ then no optimal schedule exists).

Algorithm 5

1. Draw up a list of uncompleted tasks (including also those which have yet to arrive) in nondecreasing order of d_j^*. Renumber tasks according to this order. Set $t := 0$ and $k := 0$.
2. Assign the first nonassigned active task (say T_i) from the list to the next processor. If $R_1(T_i) = 1$, set $k := k+1$. Repeat this step until sither $k = m_1$ or all processors are busy. If $k = m_1$ then go to step 3 else go to step 4.
3. Assign to free processors the nonassigned active tasks with the earliest deadlines, for which $R_1(T_i) = 0$. Go to step 5.
4. Remove a task, say T_j, which, from among all assigned tasks with $R_1(T_i) = 0$ has the latest deadline, from the processor to which it was assigned. Assign to this processor the first nonassigned active task on the list, say T_l, with resource requirement $R_1(T_l) = 1$, if for every nonassigned T_i (including T_j), $i < l$, the following condition is fulfilled $t + 1 + \lceil |B_i|/m \rceil \leq d_i^{*}$ [1],

where B_i denotes the set of nonassigned tasks with modified deadlines $\leq d_i^*$.
If this condition is not fulfilled for some i, do not change

[1] $\lceil x \rceil$ denotes the least integer not less than x.

the assignment and go to step 5. Set $k:=k+1$. If $k < m_1$ then repeat step 4.

5. Remove the assigned tasks from the list and eet $t:=t+1$ and $k:=0$. If there are any active tasks on the list then go to step 2. If there are only nonactive tasks on the list then set $t:=\min_{k\in N}\{r_k\}$, where N denotes the set of indices of these tasks and go to step 2.

The complexity of the above algorithms is dominated by that of Algorithm 4, which is $O(n^2)$. Let us note that the problem $P|res\,1\cdot 1, p_j=1, d_j|-$ may be solved without assigning modified deadlines, thus, one need only to use Algorithm 5 to solve it [5]. Taking into account the polynomial transformation $P|\beta|C_{max} \propto P|\beta|L_{max}$ [36], and Theorem 1, one may easily prove the following theorem [4].

Theorem 3

The following problems are NP-complete

1. $P2|res1\cdot\cdot, tree, p_j=1|L_{max}$,
2. $P2|res111, prec, p_j=1|L_{max}$,
3. $P3|res1\cdot\cdot, p_j=1|L_{max}$,
4. $P|res\cdot 11, p_j=1|L_{max}$,
5. $P2|res111|L_{max}$.

The complexity of other problems still remains an open question.

4. APPROXIMATION ALGORITHMS .

It follows from Section 3 that simple optimization algorithms exist only for a relatively very small group of problems. Since we are concerned with applications of scheduling algorithms in operating systems, we are mainly interested in non-enumerative algorithms which are unlikely to exist for most scheduling problems. Thus, we are justified in such cases in using approximation algorithms. In this Section we describe several heuristics. All of them concern the schedule length criterion, since until now there has been no effort made to produce and estimate approximation algorithms for other performance measures.

Given any heuristic algorithm a natural question concerns the extent to which the schedules produced by it differ from the optimum. There are three methods of evaluating such an algorithm [15]. Firstly, one can run it on selected sample problems. However, no one can prove that these problems really represent instances which may be found in practice and for which the algorithm behaves poorly. The second method consists in, using probabilistic techniques, to compare the expected values of a schedule performance measure respectively for an optimal schedule and one obtained by an approximation algorithm. It is, however, very difficult to determine a probability distribution which can be dealt with mathematically and

which also reflects the problem instances that arise in practice. Lastly, one can prove the bound on the worst case behavior of the algorithm, that is prove that the values of the schedule performance measure obtained after using the algorithm never differ from the optimal by more than some constant percentage of the optimum value. Moreover, it is worthwhile giving examples for which the approximation algorithm behaves as poorly as the bound. Such a bound is then called the best possible one [15,18].

It is last method we will be concerned with, since it provides us with a useful and precisely defined quantity using which one can compare different approximation algorithms. Moreover, when we prove a good worst-case bound for some algorithm, we know that this bound always holds for every problem instance that can exist. We will nowdescribe several heuristics for minimizing schedule length and give bounds on their woret case behavior. In this Section, for brevity we will use C to denote the schedule length.

Let us consider first the case of unit length tasks. For the independent tasks case the problem can also be thought of as multidimensional bin packing, where the number of items per bin cannot exceed the value m. Assuming $m \geq n$, we obtain a pure bin packing problem. Three main heuristics have been considered to date. They are called the first fit (FF), first fit decreasing (FFD) and level (FFL) algorithms respectively. All of them are kinds of list scheduling algorithm, that is tasks are arranged in some order on a list and at the beginning of every time slice the list is scanned from left to right and tasks which do not violate resource and precedence constraints are assigned to processors. The list scheduling algorithm is described in a more detailed way in Algorithm 6 [16].

Algorithm 6.

1. Let L be the initial list. Set $i:=1$, $L_1:=L$.
2. Set $L_i':=L_i-\{T_i:T_i$ has predecessors in $L_i\}$. Let $L_i'=<T_{i_1}, T_{i_2},\ldots,T_{i_{n_i}}>$. Assign T_{i_1} to the first processor and set $k:=1$ and $l:=1$.
3. If $T_{i_{l+1}}$ may be assigned to a processor at moment i without violating resource constraints assign it to the next free processor and set $k:=k+1$ and $l:=l+1$, otherwise set $l:=l+1$. Repeat this step until either $k=m$ or $l=n_i$
4. Set $i:=i+1$, $L_i:=L-\{T_i:T_i$ is already assigned$\}$. If L_i is a non-empty list then go to step 2, otherwise end the algorithm.

The three heuristics mentioned above differ from each other by the menner in which tasks are ordered on the initial list L. In the first fit algorithm tasks are arranged on the list in any order.

The first fit decreasing algorithm orders tasks in the nonin - creasing order of their maximal resource requirements. More clearly, for each $T_i \in L$ one can define

$$R_{max}(T_i) = \max\{(R_l(T_i)/m_l : 1 \leq l \leq s\}.$$

Then, tasks are ordered in such a way: $L=<T_{i_1},T_{i_2},\ldots,T_{i_n}>$ implies $R_{max}(T_{i_1}) \geq R_{max}(T_{i_2}) \geq \ldots \geq R_{max}(T_{i_n})$.

The level algorithm reorders tasks according ro precedence level. To describe it we need some definitions [16]. A chain of tasks in L is a sequence of tasks $T_{i_1}, T_{i_2}, \ldots, T_{i_k}$ where $T_{i_1} \prec T_{i_2} \prec \ldots \prec T_{i_k}$, and the length of this chain is k. The level of a task T_i (level (T_i)) is the length of the longest chain which begins with T_i. The level algorithm orders tasks in such a way that if $L = < T_{i_1}, T_{i_2}, \ldots, T_{i_n} >$, then level $(T_{i_1}) \geqslant$ level$(T_{i_2}) \geqslant \ldots \geqslant$ level(T_{i_n}).

Thus, to obtain a schedule one should form a list using one of the above algorithms and then apply Algoritm 6. Below we report some worst case behaviors of the heuristics described. C^* denotes the minimal schedule length, while C_{FF}, C_{FFD} and C_{FFL} denote lengths of schedules obtained after using respectively the first fit, first fit decreasing and level algorithms.

Theorem 4 [16].

For the case of independent tasks and an unbounded number of processors (i.e. $m \geqslant n$) we have:

$$1.\ \lim_{C^* \to \infty} \frac{C_{FF}}{C^*} \leqslant s + 7/10,$$

$$2.\ \lim_{C^* \to \infty} \frac{C_{FFD}}{C^*} \leqslant s + 1/3.$$

The first bound is the best possible one, while the lower bound for the second case is $s + ((s-1)/s(s+1))$.

It follows that the use of the FFD algorithm improves the worst case behavior of a schedule about 40% in terms of optimal schedule length. Let us note, that the FFL algorithm is equivalent to the FF one, since in the independent tasks case all tasks have levels equal to 1.

When the number of processors is limited , the following theorem is valid for the one resource case.

Theorem 5 [32]

$$1.\ \lim_{C^* \to \infty} \frac{C_{FF}}{C^*} \leqslant 27/10 - 24/10m\ ,$$

$$2.\ \lim_{C^* \to \infty} \frac{C_{FFD}}{C^*} \leqslant 2 - 2/m\ .$$

Moreover, the second bound is the best possible one.

Some weaker bounds also exist for the case $s > 1$ [48].

Let us now consider the case of dependent tasks. The following theorem holds.

Theorem 6 [16].

For the case of an unbounded number of processors, the following bounds exist:

$$1.\ \lim_{C^* \to \infty} \frac{C_{FF}}{C^*} \leqslant (s/2)C^* + s/2 + 1 ,$$

$$2.\ \lim_{C^* \to \infty} \frac{C_{FFD}}{C^*} \leqslant (17/10)s + 1,$$

$$3.\ \lim_{C^* \to \infty} \frac{C_{FFL}}{C^*} \leqslant (17/10)s + 1 .$$

Moreover, the first and the third bounds are the best possible ones ; the lower bound for the second case being $(1.69)s + 1$.

It follows that the worst case behavior for the FF algorithm is now considerably worse than that for the case of independent tasks. However, both the FFD and FFL algorithms significantly improve the worst case bounds.

When we limit the number of processors the only interesting bound is as follows.

Theorem 7 [16].

For $m = rsC^*$, where $r \in [0,1]$, we have

$$\lim_{C^* \to \infty} \frac{C_{FF}}{C^*} \leqslant (r - r^2/2)\, sC^* + r(s/2) + r ,$$

and this bound is the best possible one.

Next, we turn our attention to the case of arbitrary length tasks. Here the same heuristics may be used to produce schedules as for the unit execution times case, however, one should slightly modify Algorithm 6, because changes in assignment may only occur when a task finishes. While scheduling nonpreemptable tasks we then check whether enough resources are available to process an other task. When scheduling preemptable tasks it may be worthwhile preempting some unfinished task and assigning another, e.g. for FFD algorithm.

The following theorem holds for the case of independent and non-preemptable tasks.

Theorem 8 [14].

1. For an unbounded number of processors (i.e. $m \geqslant n$), we have

$$\lim_{C^* \to \infty} \frac{C_{FF}}{C^*} \leq s + 1 .$$

2. For a limited number of processors, the following bound exists

$$\lim_{C^* \to \infty} \frac{C_{FF}}{C^*} \leq \min \left\{ \frac{m+1}{2} , s+2- \frac{2s+1}{m} \right\} .$$

Moreover, these bounds are the best possible.

When scheduling independent and preemptable tasks the following theorem may be proved for the case s=1.

Theorem 9 [32]

$$1. \lim_{C^* \to \infty} \frac{C_{FF}}{C^*} \leq 3 - \frac{3}{m} ,$$

$$2. \lim_{C^* \to \infty} \frac{C_{FFD}}{C^*} \leq 3 - \frac{3}{m} .$$

These bounds are also the best possible.

When comparing the last two theorems it is quite suprising that preemption is not worthwile in two and three processor cases. Even more suprising is the fact that applying the FFD algorithm in the preemptable tasks case we may get a better schedule than that produced by the FF one.

The last theorem concerns dependent tasks.

Theorem 10 [14]

For s=1 and nonpreemptable tasks we have

$$\lim_{C^* \to \infty} \frac{C_{FF}}{C^*} \leq m ,$$

and this bound is the best possible.

There exist several other heuristics for scheduling tasks under resource constraints (for a survey we refer to [12,41]), however their worst case behaviors are not yet known.

It is also worth noting that there are approximate versions of NP-complete problems (not necessarily from scheduling theory), which theirselves are NP-complete [19,40], thus, unlikely to admit even approximation algorithms of a given quality.

5. ENUMERATIVE METHODS.

From the preceding Sections it follows that only a few polyno-

mial in time optimization or guaranteed accuracy approximation algorithms exist for scheduling under resource constraints problems. Thus, one is rather forced to use heuristics whose worst case behavior is unknown. However, there exists the computational possibility of estimating their behavior. In order to do this one should find optimal schedules using enumerative optimization methods, and then, comparing them with those produced by heuristics draw conclusions about possible improvements in these heuristics. This approach as was mentioned in Section 4, has its drawbacks, but sometimes it is the only method for estimating the efficiency of approximation algorithms. Moreover, it is possible to use enumerative methods to construct optimal schedules in some applications of computer systems especially when some task patterns are repeated and an optimal schedule may be constructed prior to its usage. It is also possible to use these methods to construct near optimal schedules, by stopping computations when a given amount of time or storage has just been exceeded.

When scheduling nonpreemptable tasks one can use either branch-and-bound or dynamic programming methods. We do not want to describe them in detail here, since both of them involve rather long expositions which also depend on the applications in which they are used. There also exist a number of articles and books dealing with these methods, for example [31]. However, one general remark may be made. Given a dynamic programming algorithm for solving any scheduling problem, an equivalent branch-and-bound algorithm can also be found.

Below we describe briefly the method for scheduling preemptable and independent tasks under arbitrary resource constraints to minimize schedule length [46]. Given the set of tasks and their resource requirements, let us number from 1 to N the resource feasible sets, i.e. those subsets of the set $T_1, T_2, \ldots, T_n$ for which the resource requirements (including processors) do not exceed any resource bound. Let Q_j denote the set of all numbers of resource feasible sets in which task T_j may be processed, and let t_i denote the duration of resource feasible set i. Then, one can formulate the linear programming problem:

Minimize

$$C_{max} = \sum_{i=1}^{N} t_i$$

Subject to

$$\sum_{i \in Q_j} t_i = p_j, \quad j=1,2,\ldots,n,$$

or in matrix notation,

$$\overline{A}\,\overline{t} = \overline{p},$$

where $\overline{A}$ is the matrix of coefficients

$$a_{ji} = \begin{cases} 1 & \text{if } i \in Q_j \\ 0 & \text{otherwise} \end{cases}$$

Of course, columns of matrix $\overline{A}$ correspond to resource feasible sets.

When applying the simplex method directly to this problem, the necessity of storage of matrix $\overline{A}$ (the largest one in the problem) greatly constrains the size of the problems which can be solved. In [46] a method was proposed, which taking advantage of the

special structure of the problem, automatically generates the consecutive columns of matrix $\bar{A}$. Thus, at every moment only one resource feasible set (which is actually checked in the simplex procedure) is stored, and thus, storage requirements are reduced to the minimum. Moreover the generation procedure is constructed in such a way, that the number of resource constraint checks is minimal.

The approach may also be applied for sets of dependent tasks. To this end, precedence constraints are presented in the form of task-on-arc graph, in which nodes (i.e. time events) are ordered in such a way that the occurrence of node i is not earlier than the occurrence of node j if $i<j$. Then, sets of tasks are found which may be processed in time intervals between the occurrence of consecutive nodes of the graph, and resource feasible sets are generated from all of these sets.

6. OTHER MODELS.

In this Section we describe other problems of scheduling under resource constraints in which other assumptions concerning the model of a computing system or goals to be achieved, are made. That is we will be concerned with scheduling in systems where each processor has its own primary memory of a given size, with the simultaneous avoidance of deadlocks and optimization of mean weighted flow time and finally with different assumptions made about the processing model.

6.1. Scheduling on processors with independent memories.

In this Paragraph we consider the model of a computer system which consists of several identical processors $P_1,P_2,\ldots,P_m$ each of which has its own memory [27,28,47]. The sizes of the memories are not identical and we denote them by $|P_1|,|P_2|,\ldots,|P_m|$. Each task T_j, $j=1,2,\ldots,m$, is characterized by its execution time p_j and memory requirement $R(T_j)$. Moreover, let F_i, $i=1,2,\ldots,m$, denote the set of all taeks which because of their memory requirements, can only be scheduled on $P_1,P_2,\ldots,P_i$, assuming that $|P_1| \geq |P_2| \geq \ldots \geq |P_m|$, and t_i denotes the sum of all the task execution times in F_i.

For such a model several heuristics which tend to minimize schedule length were proposed for scheduling nonpreemptable tasks, and their worst case behaviors were examined in [28]. Most of them are varieties of list scheduling algorithms which differ from each other by the ordering of the initial list. So, to produce schedules, we can use Algorithm 6, taking into account the fact that tasks have arbitrary processing times and another resource bound is associated with each processor. The following rules were used to order tasks (abbreviations are given in parantheses) :

1. smallest time first (STF),
2. smallest memory first, i.e. first fit increasing (FFI),
3. largest time first (LTF),
4. largest memory first, i.e. first fit decreasing (FFD).

The following theorems are valid [28] (the subscript of C denotes the scheduling algorithm producing schedules).

Theorem 11

1. For arbitrary precedence constraints, we have

$$\lim_{C^* \to \infty} \frac{C_{FF}}{C^*} \leq m.$$

2. For independent tasks, we have

$$\lim_{C^* \to \infty} \frac{C_{FF}}{C^*} \leq k + 1 + \frac{i}{2^k} \leq 1 + \log_2(m) ,$$

where k and i are chosen so that $m = 2^k + i$ and $i < 2^n$. Moreover, these bounds are the best possible.

Theorem 12

For the case of independent tasks the following bounds exist.

1. $$\lim_{C^* \to \infty} \frac{C_{STF}}{C^*} = \lim_{C^* \to \infty} \frac{C_{FFI}}{C^*} \leq k + 1 + \frac{i}{2^k} \leq 1 + \log_2(m) ,$$

where k and i are the same as in Theorem 11.

2. $$\max \left\{ \frac{C_{LTF}}{C^*} \right\} \geq \ln(m).$$

3. $$\lim_{C^* \to \infty} \frac{C_{FFD}}{C^*} \leq 2 - 1/m.$$

Bounds for cases 1 and 3 are the best possible.

It follows from the above theorems that for all of these heuristics, except the first fit decreasing one, their worst case behaviors are not much better than that of the arbitrary list scheduling algorithm. However, the first fit decreasing algorithm has quite a well-behaved bound which cannot be much improved by other heuristics involving more computations [28].

We turn our attention now to the case of preemptable tasks. In the case of independent tasks an optimization algorithm exists which may be described as follows [28].

Algorithm 7

1. Order the task list on a first fit decreasing basis.
2. Compute the values of F_i and t_i, for $1 \leq i \leq m$. Set $C_x = \max_i \{t_i/i\}$, $p_{max} = \max_j \{p_j\}$ and $C^*_{max} = \max \{C_x, p_{max}\}$. Set j=1.
3. (a) If all tasks have been scheduled, then stop. Otherwise, select the next task, T_k, from the task list and proceed to step 3 (b).

(b) If the assignment of T_k to P_j does not cause the completion time of the task to exceed C^*_{max}, then schedule T_k on P_j and return to step 3(a). If C^*_{max} is exceeded, proceed to step 3(c).

(c) Suppose the assignment of T_k to P_j causes p_a (of T_k's total processing time , p_k) to occur before C_{max} and p_b after C^*_{max}. Schedule p_a of T_k on P_j and p_b on P_{j+1} at the beginning of the schedule. Increment j by 1 and return to step 3(a).

6.2. The simultaneous liquidation of system performance failures and the minimization mean weighted flow time.

In the previous Sections it was assumed that task ready times were known a priori and that while being executed each task holds all claimed resources and releases all of them either after its completion or after being preempted. Now, let us consider the more realistic situation in which tasks from an infinite stream achieve ready status at a priori unknown moments and tasks may release some resources while holding others, then require another amounts of resources and so on.We do not assume, however, that amounts of requested and released resources as well as moments at which resource requests and releases occur are known a priori. We only assume a priori knowledge of task execution times and resource claims, i.e. maximum amounts of particular resources utilized during execution of particular tasks. Moreover, we take into account the fact that some of the resources cannot be preempted (e.g. line printers) even when the task using them is suspended. Thus, deadlock may occur, in which tasks are stuck in a circular wait, each waiting for another to release its claim on certain resources. Also, because we consider an infinite stream of tasks, certain tasks may be permanently blocked ; their resource requests may be permanently ignored. In the described situation the problem is to find scheduling and resource allocation strategies which ensure the solution of system performance failure (i.e. deadlock and permanent blocking) problems and which tend to maximize system throughput or minimize mean weighted flow time.

In majority of papers concerning the subject attention has rather been paid to the first of the aspects mentioned above. Recently, Cellary [8],[9] has presented an approach taking into account both of these aspects. He analyses a uniprocessor system with an arbitrary number of units of one nonpreemptible resource (an extension to the case of many nonpreemptible resources is given in [10]). In [8] an infinite stream of independent tasks is considered, while [9] concerns a stream of independent jobs composed of sets of dependent tasks. In the second case mean flow time of jobs instead of tasks is the system performance measure. In this approach, a deadlock avoidance method has been applied, in which safety and admission tests described by Habermann [23] have been improved by radically reducing the overhead involved. All situations which need specific servi ing, from the resource allocation point of view, have been considered and corresponding allocation algorithms have been developed for them. These algorithms utilize in a special way the SPT rule, which leads to mean flow time reduction.

The permanent blocking problem is solved by detecting permanently blocked tasks and applying of a special resource allocation

strategy which consists in the reservation of part of the resources available in the system. In opposition to Holt's solution described in [24] , this strategy does not reject in advance the possibility of granting the resource requests of tasks which do not hold any resources (in particular, resource requests of tasks just entering the system). In this way mean flow time is also reduced.

6.3. Scheduling with memory allocation.

Let us now consider a paged-virtual memory multiprocessing system (like PRIME [2]) which consists of m identical, time-shared uniprogrammed CPU's, each having a dedicated paging device, with a common primary memory of N pages and a Control Processor carrying out resource allocation. In such a system the problem occurs of scheduling tasks and allotting numbers of primary memory pages to tasks processed simultaneously in certain time intervals. System workload is based on the analytical approximation of the lifetime curve of task T_i , $e_i(s_i)$, where e_i is the mean time between succesive page faults, s_i is the number of pages allotted to task T_i, $i=1,2,\ldots,n$. Knowing $e_i(s_i)$, the mean progress rate may be determined for T_i : $f_i(s_i)=e_i(s_i)/[e_i(s_i)+t_p]$, where t_p is the page transport time.

In [13], Ferrari has proposed a memory allocation strategy following from the maximization of the total progress rate in every stated time quantum. In [45] the static case is considered, when for every task T_i from the set of $n \geq m$ independent tasks apart from $e_i(s_i)$, the size v_i is known, i.e. the number of standard instructions which have to be processed in order to complete T_i. It may be noted that the last assumption corresponds to the assumption of knowledge of task execution time in the classical deterministic scheduling problem. Then, introducing the concept of the state of a task at moment t , $x_i(t)$, which denotes the number of standard instructions processed up to moment t ($x_i(0)=0$), the function $f_i(s_i)$ may be represented by $f_i[s_i(t)] = dx_i(t)/dt$ and the performance condition for T_i by $\int_0^{C_i} f_i[s_i(t)]dt=v_i$, where C_i is the finishing time of T_i. Such a formulation allows for a thorough analysis of the properties of optimal solutions for a chosen system performance measure as well as for developing the most effective algorithms for finding these solutions. The analysis for schedule length minimization was performed in [45] for the case of independent tasks and generalized in [44] for sets of dependent tasks. In general, finding optimal memory allocation needs the solution of a constrained nonlinear programming problem. In certain cases, however, analytical results can be obtained. Also, simple approximation algorithm can be developed producing solutions not far from the optimum. While the suggested approach may seem rather sophisticated, it contains certain methodological innovations which can be utilized in more practical situations, for example when e_i depends not only on s_i and even for infinite streams of tasks. Also the consideration of other system performance measures and the interpretation in a probabilistic way of the obtained results seem to be possible. Independently of this, the above approach may be utilized for the evaluation of approximation memory allocation algorithms developed under weaker assumptions.

7. FINAL REMARKS .

In conclusion we should stress that this paper has of course not exhausted all the problems concerned with deterministic problems of computer resource allocation. In particular we have not considered issues linked with the allocation of concrete resources of one kind, like operating or mass storage. We have only examined more general problems including more types of resources, always including processors. In the basic model we have been considering it was assumed that the computing system consists of several identical processors, additional resources and a set of tasks to be processed. It was shown that the problem under consideration is difficult, so that only a few simple, optimization algorithms may be given. Thus, in practical applications one should rather tend to develop fast heuristics than to search for optimization algorithms.

Further investigation in this area should lead to the examination of the complexity of the existing open problems and to the development of new heuristics , especially for criteria other than schedule length. It is also of value to introduce new concepts concerning the processing capability of processors. For example, one can consider processors which have different speeds [42], which is sometimes assumed in papers concerning scheduling problems without additional resources [25,26,29,33,37,38]. Moreover, it is also possible to consider processors which differ from each other in their functions, e.g. front end and central processors. This leads to scheduling under resource constraints in job-shop-like systems.

We have also noted other models of computing systems, further study of which seems to be purposeful. It is worth drawing attention to the model in which tasks from an infinite stream are scheduled taking into account the nonpreemptibility of resources. This model is a good example of the utilization of the deterministic approach for describing and solving scheduling problems under quite reasonable assumptions. In a certain sense, it throws light on the "border" between probabilistic (i.e. rather dynamic) and deterministic (i.e. rather static) models. The interdigitation of probabilistic and deterministic assumptions and methods including interpretation and implementation is a general postulate for future research.

References

[1] Aho,A.V., J.E.Hopcroft, J.D.Ullman : The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass, 1974.

[2] Baskin,H.B., B.R.Borgerson, R.Roberts; PRIME - a modular architecture for terminal oriented systems, AFIPS Conf.Proc.40, JCC 1972, pp.431-437.

[3] Błażewicz,J.,: Mean flow time scheduling under resource constraints, Report No PR-19/77, Inst. of Control Eng., Tech. Univ. of Poznań, Poznań, 1977.

[4] Błażewicz,J., Scheduling with deadlines and resource constraints Report No PR-25/77, Inst.of Control Eng. Tech.Univ.of Poznań, Poznań, 1977.

[5] Błażewicz,J., Complexity of computer scheduling algorithms under resource constraints, Proc. I Meeting AFCET-SMF on Applied Mathematics, Palaiseau (France), 1978, pp.169-178.

[6] Błażewicz,J., Deadline scheduling of tasks with ready times and

resource constraints, Information Processing Letters, 1979, to appear.
[7] Bonuccelli,M., D.P.Bovet : Scheduling unit time independent tasks on dedicated resource systems, Report S-76-21, Univ. degli Studi di Pisa, Instituto di Scienze dell Informazione, 1976.
[8] Cellary,W.; Task scheduling in systems with nonpreemptible resources, in E.Gelenbe, H.Beilner (eds.), Modelling, Measuring and Performance Evaluation of Computer Systems, North Holland, Amsterdam, 1978.
[9] Cellary,W.: Scheduling dependent tasks from an infinite streams in systems with nonpreemptible resources, in Lecture Notes in Computer Science, vol.65 (G.Bracchi and P.C.Lockemann -eds.) Springer Verlag, pp.536-547.
[10] Cellary,W.: Resource allocation strategies in computer systems with nonpreemptible resources, Foundations of Control Engineering 2, No2, 1977, pp.85-92.
[11] Coffman,E.G.Jr.(ed.) : Computer and Job/Shop Scheduling Theory J.Wiley, New York, 1976.
[12] Davis,E.W.: Project scheduling under resource constraints: historical review and categorization of procedures, AIIE Transactions 5, 1973, pp.297-313.
[13] Ferrari,D.; Memory allocation in multiprocessing systems in E.Gelenbe, R.Mahl (eds.), Computer Architecture and Networks, 1974, pp.141-154.
[14] Garey,M.R., R.L.Graham : Bounds for multiprocessor scheduling with resource constraints, SIAM.J.on Computing 4, 1975, pp.187-200.
[15] Garey,M.R., R.L.Graham, D.S.Johnson; Performance guarantees for scheduling algorithms, Operations Research 26, 1978, pp.51-62.
[16] Garey,M.R., R.L.Graham, D.S.Johnson,A.C.-C.Yao : Resource constrained scheduling as generalized bin packing, J.Combinatorial Theory Ser.A 21, 1976, pp. 257-298.
[17] Garey,M.R., D.S.Johnson : Complexity results for multiprocessor scheduling under resource constraints, SIAM.J.on Computing 4, 1975, pp.397-411.
[18] Garey,M.R., D.S.Johnson; Approximation algorithms for combinatorial problems: an annotated bibliography, in J.F. Traub (ed.), Algorithms and Complexity : New Directions and Recent Results, Academic Press, New York, 1976,pp.41-52.
[19] Garey,M.R., D.S.Johnson : The complexity of near-optimal graph coloring, J.ACM 23,1976, pp. 43-49.
[20] Garey,M.R., D.S.Johnson; "Strong" NP-completeness results: motivation, examples and implications, Report "Bell Laboratories, 1977.
[21] Garey,M.R., D.S.Johnson: Computers and Intractability : a Guide to the Theory of NP-Completeness, to appear.
[22] Graham,R.L., E.L.Lawler, J.K.Lenstra, A.H.G.Rinnooy Kan,: Optimization and approximation in deterministic sequencing and scheduling theory : a survey, Report BW 82/77, Stichting Mathematisch Centrum, 1977.
[23] Habermann,A.N.,: Prevention of system deadlocks, Comm. ACM 12, No 7, 1969.
[24] Holt,R.C.: Comments on prevention of system deadlocks, Comm. ACM 14, No 1, 1971.
[25] Horvath,E.C., R.Sethi : Preemptive schedules for independent tasks, Technical Report No 162, Computer Science Dep., Pennsylvania State Univ., 1975.
[26] Ibarra,O.H., C.E.Kim: Heuristic algorithms for scheduling independent tasks on nonidentical processors, J.ACM 24, 1977,

pp.280-289.
[27] Joseph,E.C.: Innovations in heterogeneous and homogeneous distributed-function architectures, Computer, 1974, pp.17-24.
[28] Kafura,D.G., V.Y.Shen : Task scheduling on a multiprocessor system with independent memories, SIAM J.Comput. 6, 1977, pp.167-187.
[29] Kafura,D.G., V.Y.Shen : Task scheduling on processors of different speeds, Technical Report No 76-4, Dept. Comput. Sci, Iowa State University, Ames, 1975.
[30] Kariv,O., Even,S.: An $O(n^{2.5})$ Algorithm for maximum matching in general graphs, 16-th Annual Symp. on Foundation of Computer Science,IEEE, 1975, pp.100-112.
[31] Kohler,W.H., K.Steiglitz: Enumerative and Iterative Computational Approaches, Chapter 6 in [11].
[32] Krause, K.L., V.Y.Shen, H.D.Schwetman,: Analysis of several task-scheduling algorithms for a model of multiprogramming computer systems, J.ACM 22, 1975, pp.522-550 : Erratum : J. ACM 24, 1977, pp. 527.
[33] Labetoulle,J., E.L.Lawler, J.K.Lenstra, A.H.G.Rinnooy Kan : Preemptive scheduling of uniform processors subject to release dates, to appear.
[34] Lenstra,J.K.; Sequencing by Enumerative Methods, Mathematical Centre Tract 69, Amsterdam, 1976.
[35] Lenstra,J.K., A.H.G.Rinnooy Kan, : Private communication,1978.
[36] Lenstra J.K., A.H.G.Rinnooy Kan, P.Brucker : Complexity of machine scheduling problems, Ann. Discrete Math. 1. 1977. pp.343-362.
[37] Liu,J.W.S., C.L.Liu : Performance Analysis of heterogeneous multiprocessor computing systems, in E.Gelenbe and R.Mahl (eds.) Computer Architecture and Networks, North Holland, Amsterdam, 1974, pp.331-343.
[38] Liu,J.W.S., C.L.Liu,: Bounds on scheduling algorithms for heterogeneous computing systems, Proc. IFIP 74, North Holland, Amsterdam , pp.349-353.
[39] Rinnooy Kan, A.H.G., : Machine Scheduling Problems: Classification, Complexity and Computations, Nijhoff, The Hague, 1976.
[40] Sahni,S., T.Gonzales : P-complete problems and approximate solutions, Proceedings of the 15th Annual IEEE Symposium on Switching and Automata Theory, 1974,pp.28-32.
[41] Słowiński,R., : Optimal and heuristic procedures for project scheduling with multiple constrained resources - a survey, Foundations of Control Engineering 2, No1, 1977,pp.33-50.
[42] Słowiński,R.: Scheduling preemptible tasks on unrelated processors with additional resources to minimize schedule length, in Lecture Notes in Computer Science, vol.65 (G.Bracchi and R.C.Lockemann eds.) Springer Verlag, pp.536-547.
[43] Ullman,J.D., : Complexity of Sequencing Problems, Chapter 4 in [11]
[44] Węglarz,J.: Scheduling dependent tasks with memory allocation in multiprocessing systems to minimize schedule legth, Lecture Notes in Computer Science, 5.9, Modelling of Computer Systems, PP.Spies (ed.) Springer Verlag, 1977.
[45] Węglarz,J.: Multiprocessor scheduling with memory allocation - a deterministic approach, to appear.
[46] Węglarz,J., J.Błażewicz, W.Cellary, R.Słowiński : An automatic revised simplex method for constrained resource network scheduling, ACM Trans. on Mathematical Software 3,No3 , 1977.
[47] Wulf,W.A., C.G.Bell: C.mmp - a multi-mini-processor, Proc. FJCC, 1972, pp.766-778.
[48] Yao,A.C.,Scheduling unit-time tasks with limited resources, Proceedings Sagamore Computer Conference, 1974.

Performance of Computer Systems
M. Arata, A. Butrimenko, E. Gelenbe (eds.)

ANALYSIS OF A CLASS OF SCHEDULES FOR COMPUTER SYSTEMS WITH REAL TIME APPLICATIONS

A. A. Fredericks
Bell Telephone Laboratories
Holmdel, New Jersey

In this report we consider the performance analysis of a class of schedules for computer systems for which all, or part, of the resources are dedicated to real time applications. Appropriate queueing models are developed for each of the priority classes of tasks considered. The natural framework for the analysis is a multiclass priority queueing system where each class has its own (deterministic) arrival rate and (general) service time. However, it is shown that this problem can be recast as a sequence of single input (no priority) G/G/1 queueing systems. Approximate solutions are discussed, with specific results given for the case of heavy traffic.

INTRODUCTION

In this report we consider the performance analysis of a class of schedules for computer systems for which all or part of the resources are dedicated to real time applications. That is, we are concerned with systems that must respond to stimuli and perform certain tasks within specified time intervals. In addition to these "timed functions", part of the system's resources may be used to perform less critical functions.

Figure 1 shows a simplified version of such a system. The central processing unit (CPU) must scan (poll) a group of sources at a (specified) periodic rate to detect requests to transmit data. On recognizing a request, the necessary resources must be allocated (e.g., software registers, start signal hardware) and a signal sent to the source informing it to begin data transmission. The data sent by the sources must be accurately received and stored in the registers for future use. If, while performing the above "real time" functions at the desired rate, there is free CPU time, then this will be devoted to appropriate "fill" work.

For example, in a stored program control (SPC) telephone switching system, the above function might represent digit reception from a distant switching system. The "fill" work in this case might be to perform routine maintenance tasks and audits, and to respond to (teletype) requests from personnel monitoring system performance. Another example could be a computer which is responsible for data collection from other computers in a network. The "fill" work in this case might be support of a time-sharing system.

Figure 2 shows a typical "clocked" schedule for the tasks associated with our simple example system. Time is divided into slots of duration Δt. The

schedule consists of specifying which tasks are to be executed in each interval. The shaded area indicates that fill work is to be done in these slots when (and if) the scheduled tasks are finished. To complete specifications of this schedule, we must establish rules to be followed in the event that Δt elapses prior to the completion of the scheduled tasks. If task A is in progress, we will complete its execution (overrunning the clock) since it is very critical that all active sources be scanned for data transmission in the specified interval. Tasks which are treated in this manner (interrupts inhibited) are termed high priority (HP) tasks. The other scheduled tasks we label low priority (LP). Although we would like to perform these tasks at the scheduled rates, both to minimize the holding time of our resources and to provide adequate response time to new requests, it is not crucial if these desired execution times are stretched. Hence, if Δt elapses while a low priority task is executing, it will be preempted so that the more critical function of accurate data collection can be performed. The preempted task (as well as all other scheduled but unexecuted tasks in this time slot) is rescheduled in the next slot at an appropriate priority. We have indicated these priorities by the alphabetical labeling; i.e., if task D_1 is interrupted in the second slot of Figure 2, the (ordered) schedule for the next slot will be A, B, D_1.

With this introduction, we turn our attention to performance analysis questions for a general schedule of this type.

PROBLEM FORMULATION AND BASIC ASSUMPTIONS

A General Clocked Schedule

Figure 3 shows the general class of schedules we shall be concerned with. Time is divided into slots of length Δt. In each slot, there is a sequence of high priority tasks $\{H_i\}$ and low priority tasks $\{L_i\}$ scheduled. These are indicated on the figure by a (1) in the row corresponding to the indicated tasks. High priority tasks are distinguished from low priority tasks in that they run with the Δt interrupt inhibited.

If a low priority task in say the j^{th} slot is interrupted, it and all other remaining low priority work in the j^{th} slot are added to the $j + 1^{st}$ slots schedule, i.e., the remaining j^{th} work is "anded" to the $j + 1^{st}$ slot. Thus, the low priority tasks are treated with a strict preempt-resume service discipline. If all scheduled work is completed prior to the interrupt, the processor is devoted to "fill" work.

In general, clocked schedules of this type have a period for high and low priority tasks. We denote these by $\tau_h = N_h \Delta t$ and $\tau_\ell = N_\ell \Delta t$, respectively. The overall period of the schedule is thus $\tau_s = N_s \Delta t$ where N_s is the least common multiple of (N_h, N_ℓ). We will also have need to refer to the period of an

individual low priority task, L_i. This will be denoted by $\tau_{\ell_i} = N_{\ell_i}\Delta t$.*

Performance Issues and Modeling Questions

There are many performance-related issues associated with the class of clocked schedules considered here. Some are concerned with short-term variability in work loads. For example, for high priority tasks we are concerned with the frequency and extent of overruns of the Δt time slot - both of these are generally required to be quite small. One is also concerned with the probability of the total scheduled work exceeding a Δt time slot. This information is needed during the synthesis of the schedule so that the work loads in the various slots can be reasonably balanced.

These and other questions related to short-term variability generally require a detailed description of the work load in each slot. However, while this may result in difficult computations, the modeling question is clear. One needs to compute quantities associated with a known (albeit complicated distribution).

Here we are principally concerned with performance issues related to longer term variability, or more specifically, with the tails of the distributions for the delays in processing tasks. In this case, one can often make simplifying assumptions which make the analysis tractable. We next consider some of the longer term variability performance issues and construct approximate models for obtaining the relevant quantities.

High Priority Work

Let h_i^k be the work load for the i^{th} high priority task in the k^{th} slot, and let

$$w_k = \sum_{i \in I_k} h_i^k \ , \quad k=1,\ldots, N_h, N_{h+1}, \cdots$$

where I_k is the set of indices for scheduled high priority work in the k^{th} slot and $N_h\Delta t = \tau_h$ is the period for high priority work. Denoting the distribution function for w_k by F_k, we see that the short-term variability referred to above can be obtained directly from the F_k. In most cases, one can assume that the w_k, w_{k+N_h}, w_{k+2N_h}, ... are identically distributed. In what follows, we will assume that these conditions hold.

---

*For example, for the schedule of Figure 2, $\tau_S = 4\Delta t$, $\tau_A = \Delta t$, $\tau_B = 2\Delta t$, etc.

Although small, the probability of long-term delays in completing high priority work is often also of interest both for monitoring system integrity and for detecting traffic overloads. For this purpose, we see that the execution of high priority work can be modeled as a single server queueing system with deterministic interarrival times and service times taken from the (ordered) sequence of distributions

$$\{F_1, F_2, \ldots, F_{N_H}, F_1, \ldots\}$$

By using an "average" F_k we can approximate this system by a D/G/1 queueing system. The approximation methods discussed in the next section (and the Appendix) apply to such a system.

Low Priority Work

While occasional long delays in processing low priority work are generally tolerable, it is important to determine the extent of these delays. Hence, a relevant question here is to obtain the distribution function for the time from when a specific task is scheduled until it is finally completed. These delays are important not only to determine if performance objectives will be met, but also, as noted above, for monitoring machine sanity and providing direction for overload strategies.

While the delays incurred by a specific low priority task could (in principle) be obtained from a multiclass priority queueing system, the (different) deterministic interarrival times for these classes and the typical forms for the service times* generally makes this problem intractable analytically. We will show, however, that this problem can be reformulated as a standard (no priority) G/G/1 queueing system. In the next section we will apply the results of the Appendix to obtain an approximate solution.

Let j^* be the priority level of the task to be analyzed and let $\tau^* = N^*\Delta t$ be its scheduled period. We denote the processor time needed for the k^{th} scheduling of this task by s^*_k and the time required to do all higher priority work scheduled in the n^{th} $N^*\Delta t$ time period by r^*_n. That is,

$$r^*_n = \sum_{i \in I^*_n} h_i + \sum_{\substack{j \in J^*_n \\ j<j^*}} \ell_j$$

* These distributions are often discrete, taking the form $h_j k_j$ when h_j is the work time required to process one job entry in task j and k_j is the (random) number of entries found.

where I_n^*, J_n^* are the collection of indices corresponding to the high and low priority tasks scheduled in the n^{th} τ^* interval. The delays encountered by the j^{th} task are completely determined by the sequence $\{(s_n^*, r_n^*)\}$. (See Figure 4a). In fact, if we define t_n^* to be the time available in the n^{th} interval to j* work, i.e.,

$$t_n^* = \max[0, N^*\Delta t - r_n^* - v_{n-1}^*]$$

where v_{n-1}^* is the leftover work from the $n-1^{st}$ interval of higher priority than j*, then there is a complete analogy between our clocked schedule (Figure 4a) and a single server queueing system where s_n^* is the service time of the n^{th} request and t_n^* is the interarrival time between the n^{th} and $n+1^{st}$ request. (See Figure 4b). At first sight it might seem that we have lost the information regarding the actual elapsed time of the process, however, this information can be readily obtained. In the "equivalent" queueing system of Figure 4b, each "arrival" corresponds to an elapsed time interval of $\tau^* = N^*\Delta t$ units. Hence, if a request is delayed by w time units in the "equivalent" system,

$$P_d(d>w) = P_r^{\dagger}\{\text{actual delay} > W\}$$

$$= P_r\{\text{no. of arrivals during } w > \frac{W}{\tau^*}\} \qquad (1)$$

This is the delay until processing starts, the delay until completion is given by

$$P_c(c>S) = P_r\{\text{actual delay until completion} > S\}$$

$$= P_r\{\text{no. of arrivals during } w+s > \frac{S}{\tau^*}\} \qquad (2)$$

where s is the (random) service time.

Fill Work

Here we are interested in the distribution of the elapsed time for the execution of certain fill work tasks. For example, although routine maintenance is deferable, it is important to perform a certain amount of maintenance in a given time interval. For simplicity we assume that the fill work consists of repeatedly performing a single routine task (e.g., do all maintenance tasks and audits) that takes T_f time units to complete once. Thus, if t_a^i is the

---

† P_r denotes probability.

(random) amount of time available in the i^{th} period of length τ_s, then we are interested in N_f given by

$$N_f = \min\{n: \sum_{i=1}^{n} t_a^i \geq T_f\} ,$$

i.e., we have a stopping time problem.

Since, by our assumptions the t_a^i are independent, identically distributed (nonnegative) random variables, we have

$$P_f(t>T) = P\{\text{delay in accumulating } T_f \text{ real time for fill work} > T\}$$

$$= P\{\text{no. of renewals during } T_f > \frac{T}{\tau_s}\}$$

$$= P\{N(T_f) > \frac{T}{\tau_s}\} . \qquad (3)$$

The tails of this distribution can be readily estimated by using the asymptotic normality of $N(T_f)$ - we will make further use of this property shortly.

APPROXIMATE ANALYSES

Our primary interest is to develop reasonably simple analytic tools for studying the performance of clocked schedules, particularly under heavy loads. That is, we are interested in estimating the tails of the delay distribution. More specifically, if a task is scheduled every τ^* time units we are interested in the probability of execution delays exceeding $n\,\tau^*$, $n > 1$. To this end we will make extensive use of the following asymptotic approximations.

P-1). If $\tau_1, \tau_2, \ldots$ is a renewal process, and if $N(t)$ is the number of renewals that occur in a time period t, then we will assume that

$$N(t) \sim N(\bar{N}, \sigma_N)$$

where $N(.,.)$ is a Gaussian distribution,

$$\bar{N} = \frac{t}{\bar{\tau}} ; \quad \sigma_N^2 = \frac{t\sigma_\tau^2}{(\bar{\tau})^3}$$

and

$$\bar{\tau} = E\{\tau\} ; \quad \sigma_\tau^2 = \mathrm{Var}\{\tau_i\} .$$

Property 1 is, of course, asymptotically true for all renewal processes (e.g., See [1]).

P-2). If $F^c(t)$ is the complementary waiting time distribution for a G/G/1 queueing system, i.e., $F^c(t) = P_r\{\text{waiting time} > t\}$, then

$$F_c(t) = c\, e^{-at} .$$

It can be shown (see [1]) that, under rather weak conditions, this property is asymptotically true. Various methods for determining approximate values for c and a are given in [2], together with extensions to a more general class of approximation. One simple method is outlined in the Appendix. The result given there is

$$c/a = \frac{\int_0^\infty x\,dK(x)}{K(0) - \int_{-\infty}^{0} e^{ax}\,dK(x)} \tag{4}$$

where K(x) is the distribution function for $\mu = s^* - t^*$, and s^*, t^* are the service and interarrival time for the G/G/1 queueing system. The constant a is preferably determined via

$$\int_{-\infty}^{\infty} e^{az}\,dK(z) = 1 \tag{5}$$

since this yields the correct value for a in the asymptotic property, P-2 (See [1]). However, since we are primarily concerned with system behavior at loads near capacity, we can use the "heavy" traffic approximation,† (e.g. see [3])

$$a = -\frac{2\bar{\mu}}{\sigma_\mu^2} \tag{6}$$

Note, that under the heavy traffic assumptions,[3] we also have

$$K(\mu) \sim N(\bar{\mu}, \sigma_\mu) \tag{7}$$

---

†Note that we are only using heavy traffic results to estimate the exponent a. Equation 4 generally results in c < 1 and hence, we are not using a heavy traffic approximation.

Using (6) and (7) in (4) yields

$$\frac{c}{a} = \frac{\frac{\sigma_\mu}{\sqrt{2\pi}} e^{-1/2\left(\frac{-\mu}{\sigma_\mu}\right)^2} + \frac{-\mu}{2}\left[1-\mathrm{Erf}\left(\frac{-\mu}{\sqrt{2}\,\sigma_\mu}\right)\right]}{\mathrm{Erf}\left(\frac{-\mu}{\sqrt{2}\,\sigma_\mu}\right)} \tag{8}$$

where Erf(x) is the error function defined by

$$\mathrm{Erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-v^2} dr \quad .$$

Note that

$$\frac{c}{a} \underset{\mu \to 0}{\to} -\frac{\sigma^2}{2\mu} \quad ,$$

i.e., the mean delay is consistent with the heavy traffic approximation made. However, the approximation technique used here generally yields considerably better results than the heavy traffic approximation. Some accuracy comparisons are included in the Appendix, but a more detailed investigation is given in [2].

High Priority Work

The above representation readily provides an approximation for the delays for a D/G/1 system and hence applies to the high priority work discussed. Note that the deterministic structure of the input stream can be used to obtain a via (5). This will result in mean delays which are accurate, even in the low load region (e.g., see the Appendix).

Low Priority Work

Using property 1 we have that

$$P(N(w)>n) = \frac{1}{\sqrt{2\pi}} \int_{\frac{n-\frac{w}{t}}{\sqrt{w\sigma_t^2/t^3}}}^{\infty} e^{-\frac{y^2}{2}}\, dy = 1/2\left[1-\mathrm{Erf}\left(\frac{c_1 n}{\sqrt{w}} + c_2 w\right)\right] \tag{10}$$

where $N(w)$ is the number of renewals (arrivals) that occur during (a fixed) w, and $\bar{t}$, σ_t^2 are the mean and variance of the equivalent interarrival time, as defined in APPROXIMATE ANALYSIS, and

$$c_1 = \frac{\bar{t}^{-3/2}}{\sqrt{2}\,\sigma_{\bar{t}}} \quad , \quad c_2 = \frac{\bar{t}^{-1/2}}{\sqrt{2}\,\sigma_{\bar{t}}} \quad .$$

Using the approximation $F^c(w) = ce^{-aw}$ for the waiting time we get,

$$P_d(d>D) = \int_0^\infty P\left[N(w) > \frac{D}{\tau^*}\right] ace^{-aw} dw$$

which yields

$$P_d(d>D) = \frac{ca}{2\sqrt{c_2^2 + a}} \cdot \frac{\exp - 2c_1 \frac{}{\tau^*}\left[\sqrt{c_2^2 + a} - c_2\right]}{\sqrt{c_2^2 + a} - c_2} = c_3 e^{-c_4 D} \qquad (11)$$

i.e., we again have an exponential delay. This simple form allows us to continue this approximation approach to estimate $P_c(c>S) = P\{\text{delays in completing a task} > S\}$. We assume at this point that the work loads for the j^{*th} task are given by

$$d_{j^*} = b_{j^*} k_{j^*}$$

where b_{j^*} is the (fixed) work time for job entry in task j^*, and k_{j^*} is the random number of entries found with distribution

$$k_{j^*} \sim P(k_{j^*}) \quad .$$

It is then easy to show (by conditioning on the number of job entries) that the desired distribution take the form

$$P_c(c>S) = \sum_{k_{j^*}=1}^{\infty} A_{k_{j^*}} P_{k_{j^*}} \qquad (12)$$

where

$$A_{k_{j^*}} = A_{k_{j^*}}(S,\bar{t},\bar{s},\sigma_t,\sigma_s,\mathrm{Erf}(\cdot))$$

i.e., the $A_{k_{j^*}}$ depend only on the parameters indicated, and involve expressions no more complicated than the error function.

While using means and variances to characterize the work of priority higher than j^* is often adequate (since we are usually combining many tasks), it is sometimes necessary to have a better characterization of the service times for j^*.

Fill Work

Here we wish to compute

$$P_f(f>T) = P\left[N(T) > \frac{T_f}{\tau_s}\right]$$

where T and τ_s are fixed. Thus, Equation 10 with the suitable parameters applies, i.e.,

$$P_f(f>T) = 1/2\left[1-\mathrm{Erf}\left(\frac{c_1' n'}{\sqrt{T}} + c_2'\sqrt{T}\right)\right] \qquad (13)$$

where now

$$c_1' = \frac{\bar{t}_a^{-3/2}}{\sqrt{2}\,\sigma_{t_a}} \quad ; \quad c_2' = \frac{\bar{t}_a^{-1/2}}{\sqrt{2}\,\sigma_{t_a}} \quad ; \quad n' = \frac{T_f}{\tau_s}$$

Often, the fill work consists of scanning a table for requests and reacting to these by loading work entries in the various scheduled tasks. In this case we are interested in the response time to a request loaded at an arbitrary point in the table, i.e., the forward recurrence time of the table scan delays. This is readily computed to be

$$P(r>R) = \int_R^\infty P_f(f>T)\,\frac{dT}{m}$$

$$= \left[\frac{1}{\sqrt{2\pi}} \int_{\frac{R-\bar{t}}{\sigma_t}}^{\infty} e^{-t^2/2}dt\right]\left(\frac{\bar{t}-R}{m}\right) + \frac{\sigma_t e^{-1/2\left(\frac{R-\bar{t}}{\sigma_t}\right)}}{m\sqrt{2\pi}} \qquad (14)$$

where

$$\bar{t} = \frac{T_f T_s}{\bar{t}_a}\ ,\ \sigma_t^2 = \frac{T_f \sigma_{t_a}^2}{t_a^3}\ \tau_s^2\ ,\ \text{and}$$

where m, the mean time to complete once pass of fill work is given by

$$m = \int_0^\infty y dP_f(y) = \frac{\sigma_t}{\sqrt{2\pi}}\ e^{-\left(\frac{\bar{t}}{\sigma_x}\right)^2/2} + \frac{\bar{t}}{\sqrt{2\pi}} \int_{\bar{t}/\sigma_t}^{\infty} e^{-t^{-2}/2}dt$$

References

|1| W. Feller: An Introduction to Probability Theory and Its Application, John Wiley and Sons, N.Y., N.Y. (1966, 1971).

|2| A. A. Fredericks: A Class of Approximations for Delay Distributions in a G/G/1 Queueing System, to be published.

|3| J. F. C. Kingman: The Heavy Traffic Approximation in the Theory of Queues, in W. L. Smith, R. I. Wilkinson, eds., Proceedings of the Symposium on Congestion Theory, University of North Carolina (Chapel Hill), 137-169 (1964).

APPENDIX

In [2] we consider a class of approximations to the delay distribution for a G/G/1 queueing system. These approximations apply to delay distributions which admit the expansion $W(x) = \sum_{i=0}^{\infty} c_i e^{-a_i x}$. We outline below the nature of these approximations for the simplest case.

We wish to obtain an approximate solution for the waiting time distribution W(x) for a G/G/1 queueing system. That is, we have a single server queue where the interarrival times, t, between customers are independent and identically distributed, taken from a general distribution, A(t). The service times, s, are independently drawn from an arbitrary distribution B(s). A first in-first out discipline is assumed. If we let u = s-t and denote its distribution by K(u), then W(x) satisfies the well known Lindley integral equation

$$W(x) = \int_{-\infty}^{x} w(x-y)\, dK(y) \qquad \text{A-1}$$

We wish to approximate W(x) by an exponential distribution, i.e.,

$$W(x) = 1 - ce^{-ax} \qquad \text{A-2}$$

where c, a are suitable chosen.

Now if W(x) were exponential, then A-1 implies

$$1-ce^{-ax} = \int_{-\infty}^{x} \left(1-ce^{-a(x-y)}\right) dK(y) = K(x) - ce^{-ax} \int_{-\infty}^{x} e^{ay} dK(y). \qquad \text{A-3}$$

While, in general, this equation cannot be valid for all x, unless W(x) is exponential, we note that if we require equality to hold in the limit as $x \to \infty$ we obtain the requirement

$$\int_{-\infty}^{\infty} e^{ay} dK(y) = 1 \qquad \text{A-4}$$

This is a well known relationship which, if satisfied for some nonzero a, implies that (see [1])

$$1 - W(x) \underset{x\to\infty}{\to} ce^{-ax} \qquad \text{A-5}$$

Thus, if an exponential form is to be chosen, this is a logical choice for the exponent, a.

To determine c, we match the first moment of the integral Equation A-1 for the desired exponential form. That is, for the exact W(x) we should have from A-1 that

$$\int_0^\infty x dW(x) = \int_0^\infty x \left[W(0)dK(x) + \int_{-\infty}^{x} d_x W(x-y)dK(y) \right] . \qquad \text{A-6}$$

Substituting $W(x) = 1 - ce^{-ax}$ into A-6 yields the relation (after some algebra)

$$\frac{c}{a} = \frac{\int_0^\infty x dK(x)}{K(0) - \int_{-\infty}^{0} e^{az}\, dK(z)} \qquad \text{A-7}$$

Using A-2 with a given by A-4 and c given by A-7 is one of a class of approximations for delay distributions for G/G/1 systems considered in [2]. Alternate methods for obtaining a (and c), suitable for given circumstances, and for increasing the order of the approximation, are given there. One simple choice for a in complex situations is just the heavy traffic approximation value

$$a = \frac{-2\bar{u}}{\sigma_u^2} \qquad \text{A-8}$$

To compare the accuracy of these approximations, we consider the special case of an M/D/1 system. For this case, A-7 reduces to

$$\frac{c}{a} = \frac{\bar{d}(\rho-(1-e^{-\rho}))(\rho+a\bar{d})}{\rho e^{-\rho a\bar{d}}} \qquad \text{A-9}$$

where d is the (constant) service time and $\rho = \bar{d}/\bar{t}$.

The following table shows the excellent agreement that is obtained using A-9 with a given by A-4, as well as the loss in accuracy that accompanies simpler approximation.

Accuracy of Approximations

Mean Delays

ρ	True	A_1	A_2	A_3	A_4
.01	.00505	.00504	.0051	12.5	50.5
.1	.0556	.0554	.083	1.54	5.55
.5	.500	.495	.704	1.03	2.00
.9	4.50	4.50	5.03	4.90	5.56
.99	49.5	49.5	50.1	50.4	50.5

A_1 - Using A-9, a given by A-4

A_2 - Using A-9, a given by A-8

A_3 - Using A-7, and K Gaussian (Eq. (8) of Text)

A_4 - "Heavy Traffic" mean, i.e., $\frac{\sigma_u^2}{2\bar{u}}$

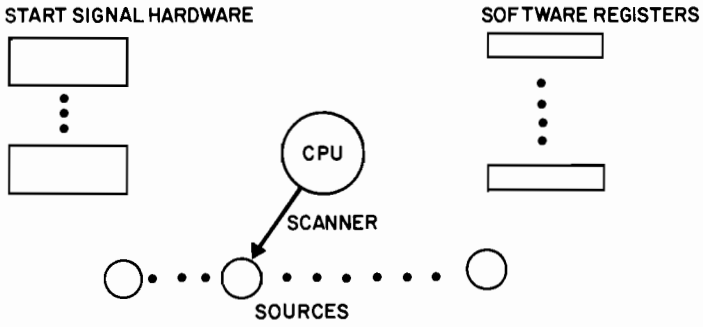


FIGURE1 EXAMPLE SYSTEM

TASK	PRIORITY
SCAN ACTIVE SOURCES FOR DATA	A
ASSIGN REGISTER TO NEW REQUEST	B
SEND START SIGNAL	C
SCAN FIRST HALF OF SOURCES FOR REQUESTS	D_1
SCAN SECOND HALF OF SOURCES FOR REQUESTS	D_2
FILL WORK	

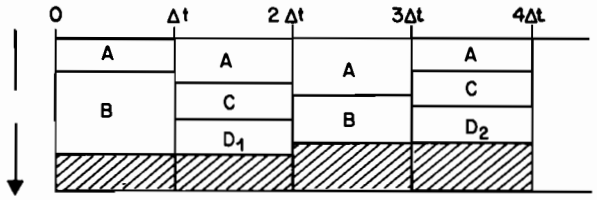


FIGURE 2 CLOCKED SCHEDULE

TASK 0	Δt	2Δt	3Δt	4Δt TIME
H_1	1	0	1	0
H_n	1	0	0	1
L_1	0	1	0	1
L_m	0	0	1	0
FILL	1	1	1	1

FIGURE 3 GENERAL CLOCKED SCHEDULE

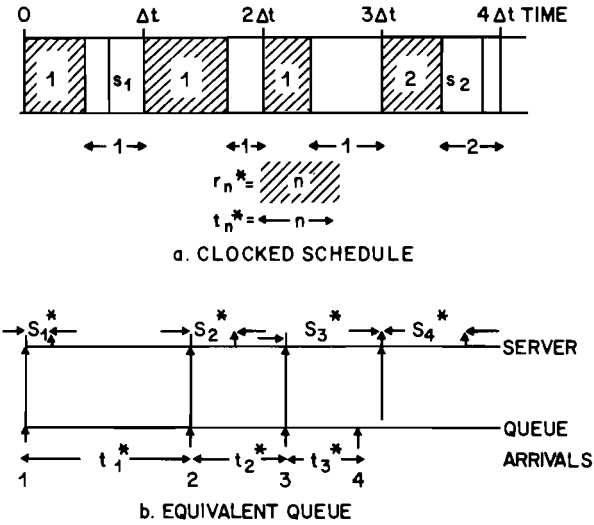


FIGURE 4 CLOCKED SCHEDULE AND EQUIVALENT QUEUE

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

A LOAD-SENSITIVE SCHEDULER FOR INTERACTIVE SYSTEMS

Manfred Ruschitzka
Department of Computer Science
Rutgers University
New Brunswick, New Jersey

Brinch Hansen defined the highest response-ratio next discipline [1] to implement the policy of equitable sharing in batch processing environments. We present a novel scheduling scheme, the processor-sharing common service-ratio (CSR) algorithm, which achieves this policy in interactive systems. The priority of a job under this scheme is defined by the ratio between the time it spent in the system and its attained service. The analysis of this algorithm for arbitrary service time distributions and Poisson arrivals shows that the average response time of a job conditioned on its service requirement equals that of a job under the round-robin scheme. However, under heavy load, the CSR algorithm resembles the feedback scheme with respect to the favorable treatment of short jobs versus long jobs. It thus strikes a balance between the round-robin and the feedback scheme as a function of the current system load. In view of these characteristics and the simplicity of its specification, the CSR algorithm offers itself as an attractive alternative to these two standard interactive scheduling algorithms.

INTRODUCTION

In a multiprogramming system, an individual user views the processing rate of a job as the ratio between the service requirement and the response time. This externally observable rate, sometimes called the virtual processing rate, is the time average of the instantaneous processing rates which the job experiences during its lifetime in the system. The instantaneous processing rates are, of course, a function of the scheduling algorithm and the system load. It has been argued that all jobs should be scheduled in such a way as to keep their average virtual processing rates the same; this policy has been called equitable sharing [1]. It is well known that both the processor-sharing round-robin (RR) and the preemptive last-come first-serve (LCFS) algorithms achieve equitable sharing [4]. For batch processing systems, Brinch Hansen defined the nonpreemptive highest response-ratio next (HRN) algorithm [1] which attempts to satisfy this policy by striking a balance between the shortest-job-first (SJF) and the first-come first-serve (FCFS) algorithms. Under HRN, the priority of a job is defined by the ratio between the time it has spent in the system and its service requirement. The author has analyzed this algorithm for arbitrary service time distributions and Poisson job arrivals [10]. There are two reasons why the HRN scheme is not suitable for interactive environments: the algorithm is nonpreemptive and requires knowledge about the service requirement of a job at the time of its arrival. In this paper, we present a novel algorithm, the processor-sharing common service-ratio (CSR) algorithm, which is related to the HRN scheme, but overcomes its drawbacks regarding interactive systems.

THE PROCESSOR-SHARING COMMON SERVICE-RATIO ALGORITHM

According to the classification scheme of Ruschitzka and Fabry [9], a scheduling algorithm is specified in terms of a priority function of an arbitrary number of job and system parameters, a decision mode, and an arbitration rule.

The decision mode specifies at what instants in time the priority function is to be evaluated for all resident jobs and the server is to be allocated to the job(s) with the highest priority (the arbitration rule resolves conflicts). The priority function of the CSR scheme is defined as

$$P(r,a) = r/a \qquad (1)$$

where the job parameters r and a are the (real) time a job has spent in the system and its attained service respectively. Since $a \leq r$, we have $P(r,a) \geq 1$. The decision mode is processor-sharing, i.e. scheduling decisions are made continuously. (This mode may be viewed as the limiting case of a quantum-oriented mode with the quantum size approaching zero.) The arbitration rule is sharing, i.e. all jobs with the same highest priority are serviced simultaneously, generally at different rates.

To illustrate the internal behavior of the CSR algorithm, assume that there is only one job in the system and that it has attained A seconds of service during R seconds in the system. By definition, its priority is $R/A \geq 1$. Assume also that a new arrival occurs. Initially, its priority will be zero divided by zero and, thus, undefined. If it does not get serviced immediately, however, its priority will jump to infinity since it will have been in the system for a finite amount of time without having obtained any service. Thus, the scheduler will immediately service the new arrival until its priority equals that of the original job. This is achieved within an infinitesimal interval of time of which the new arrival attains the fraction A/R in service. Thereafter, both jobs have the same (highest) priority in the system and will be serviced simultaneously. As the name implies, the CSR scheme always assures that a single priority value (ratio of time in system to attained service) is common to all resident jobs. Figure 1 illustrates this characteristic in terms of a real-time/service-time diagram. In such a diagram, a job is represented by a point. Upon arrival, it appears in the origin and proceeds to move simultaneously upward at unit rate and to the right at its service rate. At any instant in time, its priority is given by the tangent of the radial on which it resides. Since the CSR scheme maintains

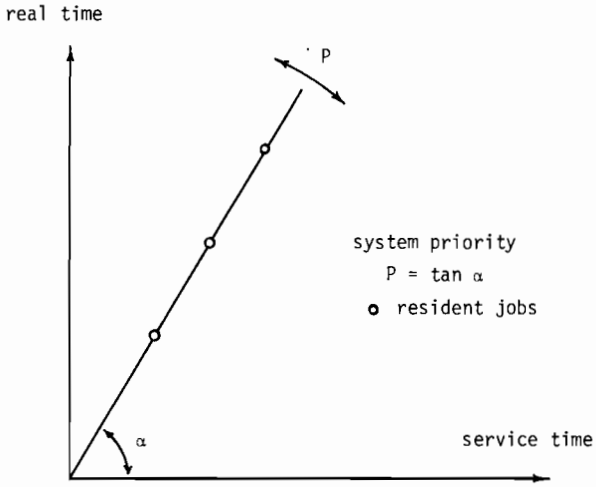


Figure 1. Resident jobs are positioned on a single radial.

equal priorities for all resident jobs, all jobs reside on a single, common radial. This radial moves upward and downward as a function of the current system load. Its tangent at any instant in time will be called the system priority P.

We now relate the instantaneous processing rates of the resident jobs to the system priority P. Let r_k, a_k, and P_k denote the time spent in the system, the attained service, and the priority of a job with index k respectively at some instant in time. Assume that there are m jobs in the system, i.e. $1 \le k \le m$. After an infinitesimal interval dt, the system priority will have increased by $dP=dP_k$, the attained service of job k by da_k, and its time in the system by dt. Thus, the priority of job k will equal

$$P + dP = (r_k + dt)/(a_k + da_k). \tag{2}$$

Expressing P by r_k/a_k, multiplying the equation by $a_k(a_k+da_k)/dt$, and taking the limit $dt \to 0$ yield an expression for the processing rates $da_k/dt=a_k'$,

$$a_k' = (1 - P'a_k)/P, \tag{3}$$

where P'=dP/dt. Clearly, the sum of all processing rates must equal one. Thus, summing over k and multiplying by P yields an expression for the derivative of the system priority:

$$P' = (m - P)/\sum_{k=1}^{m} a_k. \tag{4}$$

Note that the system priority will stay constant for P=m. Otherwise, it converges toward m. Substituting equation (4) in equation (3) yields

$$a_k' = [1 - (m - P)a_k/\sum_{k=1}^{m} a_k]/P. \tag{5}$$

This equation for the processing rates demonstrates one of the most desirable characteristics of the CSR algorithm. When the system load increases due to the arrival of new jobs (m>P), the service rate of a job is monotonic decreasing with its attained service. A job with a very large amount of attained service will therefore receive almost no service. In other words, the algorithm favors the potentially short jobs when the load increases. Conversely, when the load diminishes due to departures (m<P), the service rate of a job increases monotonically with its attained service. Thus, the algorithm favors the long background jobs when the load goes down and satisfactory service for short jobs is a priori assured. This load-sensitive behavior of the CSR algorithm is its main advantage over the RR algorithm which services all resident jobs at the same rate independent of their attained service.

To characterize the fluctuations of the system priority P, it is advantageous to introduce a fictitious job, called the center job c, which represents an average of the resident jobs. The attained service a_c of the center job is defined as the average of the attained services of the m resident jobs:

$$a_c = \sum_{k=1}^{m} a_k/m. \tag{6}$$

Its residence time is defined to equal the system priority multiplied by its attained service. This definition assures that the center job will always share the same radial with the resident jobs in the real-time/service-time diagram. It follows from equation (5) that the center job is serviced at rate 1/m. Between

arrival and departure epochs, i.e. when m is fixed, the center job therefore provides a simple means for determining the movement of the system radial. Figure 2 illustrates the system behavior under increasing system load (P<m=2). During intervals of length I, the center job attains I/m seconds of service, thus fixing the new position of the system radial. The positions of the resident jobs are then determined by this radial. Note the preferential treatment of the job with less attained service. Figure 3 depicts the case of decreasing system load

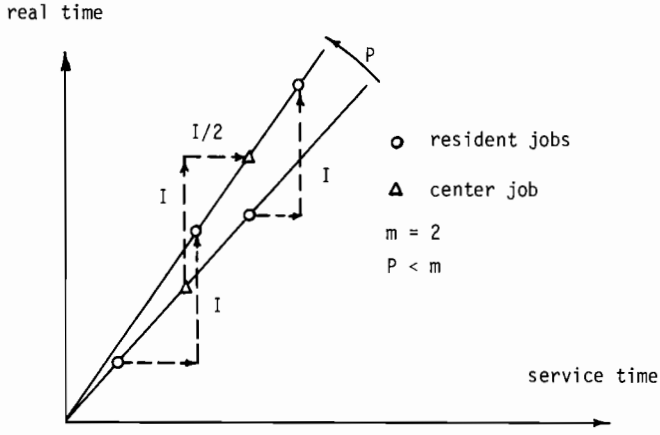


Figure 2. Service rates under increasing system load.

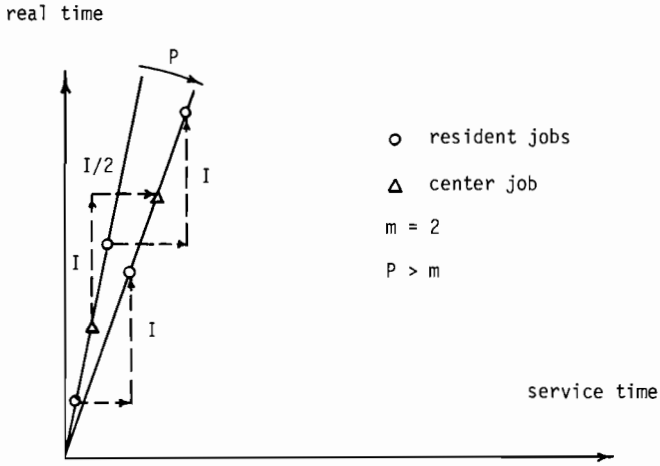


Figure 3. Service rates under decreasing system load.

(P>m=2) and the preferential treatment of long jobs. Equation (5) indicates that the discrimination between short and long jobs is more pronounced when the difference between their attained services is large. This characteristic holds under both increasing and decreasing system loads. We now proceed with the analytical derivation of the response characteristics of the CSR algorithm.

THE MODEL

We shall use the response function, the average time a job spends in the system conditioned on its service requirement, to describe the behavior of a system under the CSR algorithm in equilibrium. The system is modeled by an M/G/1 queuing system, i.e. a system with Poisson job arrivals, a general (arbitrary) distribution of job service times which are mutually independent, and one server. In addition to the processor-sharing mode, we assume that the system is work-conserving, i.e. that preemption and resumption of a job will not require additional service time. In sum, we model a time-sharing computer facility as a work-conserving, processor-sharing M/G/1 queuing system. When a job arrives at this system, it adds its service time to the unfinished work [4], the work (in units of service time) which the system has yet to perform on resident jobs. Periods during which the unfinished work is positive are called busy periods.

We begin to observe the system at the beginning of an arbitrary busy period at time zero and we number jobs in the order of their arrival with index j (j=0,1,2, ...). Their arrival epochs and service time requirements will be denoted by random variables T_j and S_j respectively (T_0=0). I_j stands for the time between the arrivals of jobs j-1 and j, i.e. $I_j=T_j-T_{j-1}$ (j=1,2, ...). We express delays and other random variables of interest for a particular (but arbitrary) job n, and refer to this job as the test job. Introducing special symbols for its arrival epoch $t=T_n$ and its service requirement $x=S_n$, we denote its residence time in the system (the time between arrival and departure) by R(t,x). This residence time consists of the test job's service time x and its waiting time W(t,x):

$$R(t,x) = W(t,x) + x. \tag{7}$$

It is advantageous to relate the waiting time of a job to the amount of service (work) performed on other jobs. Using this approach [4,10,11], we distinguish between two groups of jobs: the early arrivals (j<n) which arrive before the test job and the late arrivals (j>n) which arrive after it. We define the early work $V_1(t,x)$ (late work $V_2(t,x)$) as the total service performed on all early (late) arrivals during the test job's residence time. Clearly, an early arrival which departed prior to the test job's arrival at time t cannot benefit from the early work. Similarly, a job arriving after the test job's departure cannot contribute to the late work. With these definitions, equation (7) may be rewritten as

$$R(t,x) = V_1(t,x) + V_2(t,x) + x. \tag{8}$$

Note that the residence time, the early work, and the late work are stochastic processes defined for any job n (n=0,1,2, ...) arriving at time $t=T_n$. Equilibrium values are obtained by letting n (and, thus, t) approach infinity.

The response function R(x) can now be expressed as the time average of the residence time in equation (8). With the definition of the time average Q(x) of a stochastic process Q(t,x) with a constant parameter x,

$$Q(x) = \lim_{T\to\infty} (1/T) \int_0^T E[Q(t,x)]\, dt, \tag{9}$$

we obtain

$$R(x) = V_1(x) + V_2(x) + x. \tag{10}$$

λ will denote the job arrival rate. The random variable S and the functions G and g stand for the service time requirement of a job, the service time distribution function and its probability density function respectively. G^c denotes the complement of G: $G^c(x)=1-G(x)$. The truncated service time is defined by $S_{<x} \equiv \min[S,x]$. Its m-th moment $ES^m_{<x}$ equals

$$ES^m_{<x} = \int_0^x s^m g(s)ds + x^m G^c(x) = \int_0^x ms^{m-1} G^c(s)ds \tag{11}$$

where the first expression can be derived from the last one via an integration by parts. By definition, $ES^1_{<x} = ES_{<x}$ and $ES^m_{<x}$ approaches ES^m for $x\to\infty$.

THE EARLY-WORK PROCESS

The early work is the amount of service which early arrivals (jobs $j<n$) attain during the test job's residence. Equivalently, it may be defined as the service which early arrivals attain prior to the test job's departure minus the service they attain prior to its arrival. There are two types of early arrivals: those which are serviced to completion and exit before the test job departs, and those which are still resident at the test job's departure epoch. We consider the latter type first.

As noted earlier, the priorities of all jobs are equal. In particular, the priority of a resident early arrival and that of the test job are equal at the departure epoch of the test job. Thus, with a_j denoting the attained service of a resident job j, and using $L=R(t,x)$ for the residence time of the test job, we have

$$P(L,x) \equiv L/x = (t-T_j + L)/a_j \equiv P(t-T_j + L, a_j).$$

Note that $T_j<t=T_n$ since $j<n$. Solving for a_j, we obtain

$$a_j = x(t-T_j + L)/L. \tag{12}$$

Equation (12) specifies the attained service a_j of a resident early arrival at the test job's departure epoch. Early arrivals of the other type will already have departed after receiving their service requirement S_j. However, had the service requirement S_j of such a departed early arrival been larger than a_j, this job would still be resident with exactly a_j seconds of attained service. Therefore, any early arrival will have attained the minimum of S_j and a_j respectively when the test job departs. The total amount of the early work which is to be performed during the test job's residence time can now be expressed as the sum of the individual minima minus the service performed prior to the test job's arrival at epoch t, or t-H, where H denotes the sum of the durations of all idle

periods preceding t. Thus, we have

$$V_1(t,x) = \sum_{j=0}^{n-1} \min[S_j, x(t-T_j+L)/L] - (t-H). \tag{13}$$

By expressing t and T_j as sums of interarrival times and subtracting S_j from the minima we get

$$V_1(t,x) = \sum_{j=0}^{n-1} \min[0, x(t-T_j+L)/L-S_j] + \sum_{j=0}^{n-1} (S_j-I_{j+1}) + H. \tag{14}$$

The last two terms of equation (14) represent an expression for the unfinished work U(t). Using the identity min[0,e]=-max[0,-e], we rewrite the first term, which we name -Z(t,x), so that

$$Z(t,x) = \sum_{j=0}^{n-1} \max[0, S_j-x(t-T_j+L)/L]. \tag{15}$$

Thus, equation (14) for the early work becomes

$$V_1(t,x) = U(t) - Z(t,x). \tag{16}$$

We now form the time average of the early work as the difference of the time averages for U(t) and Z(t,x), i.e. $V_1(x)=U-Z(x)$. Substituting the well-known expression for U [4], we have

$$V_1(x) = \lambda ES^2/2(1-\rho) - Z(x) \tag{17}$$

where $\rho = \lambda ES$ denotes the system load. It remains to determine Z(x).

The time average Z(x) is defined by equation (9):

$$Z(x) = \lim_{T\to\infty} (1/T) \int_0^T EZ(t,x)\, dt. \tag{18}$$

As indicated by equation (15), the process Z(t,x) is a sum of contributions from all jobs preceding the test job. Figure 4 illustrates such a contribution from a job j<n (which arrives at time $\xi=T_j$) as a function of t, the arrival time of the test job. For Poisson arrivals, the contributions of the various arrivals to EZ(t,x) are equal to the integral of the expected contribution of a job arriving at epoch ξ times the probability $\lambda d\xi$ of the occurrence of such an arrival. From the definition of Z(t,x) in equation (15) we get

$$EZ(t,x) = \int_0^t \int_{x(t-\xi+L)/L}^{\infty} [s-x(t-\xi+L)/L]\, g(s)\, ds\, \lambda d\xi.$$

Substituting $y=x(t-\xi+L)/L$ and evaluating the inner integral yield

$$EZ(t,x) = \lambda(L/x) \int_x^{x+xt/L} (ES-ES_{<y})\, dy. \tag{19}$$

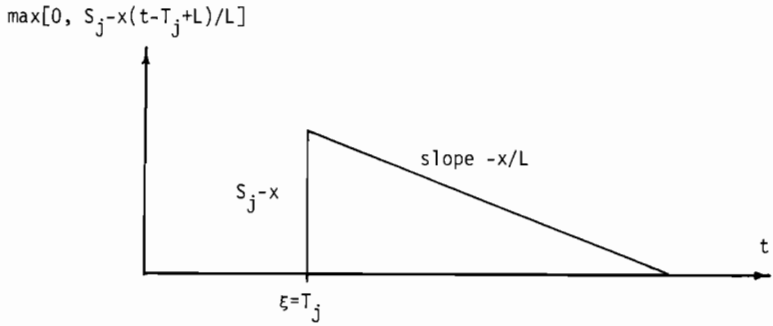


Figure 4. The contribution of job j to the process Z(t,x).

With this expression, the time average Z(x) from equation (18) becomes

$$Z(x) = \lim_{T\to\infty} (\lambda/Tx) \int_0^T \int_x^{x+xt/L} (ES-ES_{<y}) \, dy \, dt.$$

Note that L=R(t,x) is a function of t. Denoting the inverse of xt/L by f(y,t,x) and exchanging the order of integration yield

$$Z(x) = \lim_{T\to\infty} (\lambda/x) \int_x^{x+T} (ES-ES_{<y}) \int_{f(y,t,x)}^{T} (L/T) \, dt \, dy.$$

The inverse function f(y,t,x) may have multiple values, but under the limit, the lower integration boundary of the inner integral may be replaced by any finite value, e.g. zero. More importantly, the limit of the inner integral constitutes the Cesàro mean of the test job's residence time which we may denote by R(x). Thus, after taking the limit we obtain

$$Z(x) = (\lambda/x)R(x) \int_x^{\infty} (ES-ES_{<y}) \, dy$$

and an integration by parts results in the final expression

$$Z(x) = \lambda R(x)[(ES^2-ES^2_{<x})/2x - (ES-ES_{<x})] \qquad (20)$$

for the time average Z(x).

THE LATE-WORK PROCESS

The late work is the amount of service which jobs arriving after the test job will attain during the test job's residence time. Together with the early work, it represents the delay encountered by the test job. Similar to the approach used to determine the early work, we shall relate job priorities at the departure epoch of the test job in order to determine the late work. Assuming

again that the test job n remains in the system for L seconds and that its service requirement is $x=S_n$, we observe a job $j>n$ which arrives i seconds after the test job ($i<L$) and attains A seconds of service by the time the test job departs. Equating the priorities of the test job and a resident late arrival,

$$P(L,x) \equiv L/x = (L-i)/A \equiv P(L-i,A),$$

we obtain the amount of attained service as

$$A = x(1 - i/L). \tag{21}$$

Departures of late arrivals are handled like those of early arrivals. If a late arrival departs before the test job, its service requirement S must have been less than the value of A in equation (21). Thus, any late arrival will contribute min[S,A] to the late work, provided it arrives during the test job's lifetime L. Making use of Poisson arrivals, the expectation of the sum of the contributions of the late arrivals may again be replaced by an integral. The time average for the late work, conditioned on x and L, therefore amounts to

$$V_2(x,L) = \int_0^L E[\min[S,x(1-i/L)]]\ \lambda\ di.$$

Expressing the expectation of the minimum in terms of the expectation of the truncated service time, we have

$$V_2(x,L) = \lambda \int_0^L ES_{<x(1-i/L)}\ di.$$

Substituting y for $x(1-i/L)$ and integrating by parts yield

$$V_2(x,L) = \lambda\ L\ [ES_{<x} - ES^2_{<x}/2x].$$

After unconditioning with respect to L and using R(x) for the time average of L, we obtain

$$V_2(x) = \lambda\ R(x)\ [ES_{<x} - ES^2_{<x}/2x] \tag{22}$$

for the final expression for the time average of the late work.

PROPERTIES OF THE RESPONSE FUNCTION

With the time averages for the early and the late work established, we rewrite equation (10) by substituting equations (17), (20), and (22):

$$R(x) = \lambda\ ES^2/2(1-\rho) - \lambda\ R(x)[ES^2/2x - ES] + x.$$

Solving for R(x) and using the symbol $\rho=\lambda\ ES$ for the system load, we now obtain the appealingly simple expression for the response function:

$$R(x) = x/(1-\rho). \tag{23}$$

This result is valid for work-conserving, processor-sharing M/G/1 systems under the CSR algorithm, i.e. under a priority function $P(r,a)=r/a$. Unconditioning with respect to the service requirement x yields the overall average residence

time $R=ES/(1-\rho)$, and Little's theorem [5] provides the average number N of jobs in the system: $N=\rho/(1-\rho)$. The form of the CSR response function is quite familiar; it equals those of the processor-sharing RR and the preemptive LCFS algorithms.

Kleinrock [4] has summarized four appealing properties of such a linear response function with $R(0)=0$. First, the linearity assures a fair allotment of the processor to the competing jobs, thus avoiding job discrimination. On the average, the time a job spends in the system is proportional to its service requirement. Second, the average response time is independent of the shape of the service time distribution; it depends only on the first moment of the service time distribution and the arrival rate. Thus, concerns about higher moments, like under FCFS where the average delay is proportional to the second moment, need not be dealt with. Third, the fairness of the algorithm removes the motivation for users to "beat" the system via countermeasures [3] like splitting jobs to reduce response time. Such countermeasures tend to increase overhead and, thus, cause a deterioration of the overall system behavior. Fourth, in comparison with the response function $U+x$ for FCFS, short jobs are serviced more rapidly than long ones. In an interactive system, this preference is crucial from a psychological point of view [6].

While the averages of the observed virtual processing rates are identical under the RR and the CSR scheme, the CSR scheme differs from the RR scheme in its load-sensitive determination of the instantaneous processing rates. It has been shown that jobs with large amounts of attained service are treated like background jobs when the system load increases and are favored when the load goes down. The opposite is true for short jobs. Equation (5) shows that the service rates of all jobs will only be equal when the system priority equals the number of resident jobs; in this case the CSR scheme acts like the RR scheme. In comparison with the feedback algorithm [4] which is most discriminating with respect to treating long jobs as background jobs, the CSR algorithm is superior in the sense that long jobs are never completely denied service for any length of time. Such denial of service is particularly disturbing in an interactive setting where users will eventually abort the job, thus throwing away the attained service [8]. In view of these properties, the CSR algorithm represents an attractive compromise between the RR and feedback algorithms.

CONCLUSION

A variety of scheduling algorithms implements the policy of equitable sharing, including the nonpreemptive highest response-ratio next (HRN) [1,10], the preemptive last-come first-serve (LCFS) [4], and the processor-sharing round-robin (RR) [4] algorithms. We defined a novel equitable-sharing algorithm, the processor-sharing common service-ratio (CSR) algorithm, in which the priority of a job is defined by the ratio of the time it spent in the system and its attained service. The internal, load-sensitive behavior of this algorithm was discussed and two expressions for the instantaneous rates at which jobs are serviced were derived. We then analyzed the CSR algorithm for arbitrary service time distributions under the assumption of Poisson job arrivals. Our main result, equation (23), establishes the linearity of the response function, the equilibrium average of a job's response time conditioned on its service requirement, for the CSR algorithm. Thus, the first moments of the response are shown to be identical to those of the RR and the LCFS algorithms. However, the internal behavior of the CSR scheduler shows that the service rates of individual jobs vary with their attained service as well as with the system load. In particular, long jobs are discriminated against when the system load increases, but favored when the system is unloaded. Such a discrimination is typical of the feedback algorithm, but the CSR scheme avoids its drawback with respect to the denial of service to long jobs under a heavy load. Thus, the CSR algorithm strikes a balance between the RR and the feedback algorithms. While achieving the same average response times as the RR scheme, its load sensitivity allows for a preferential treatment of short jobs

under increasing loads.

It is known that under certain circumstances the distribution of the state (e.g. the number of jobs) in a computer network can be expressed as the product of the state distributions of the individual nodes. In particular, this is true if job service times have a rational Laplace transform and if the nodes are scheduled via the RR of LCFS algorithms [7]. Since the CSR algorithm displays the same response average as those two algorithms, and since it satisfies the immediate service criterion [2], its network behavior should be analyzed. The validity of the product form for the CSR algorithm would broaden the scope of applicability for many network results. In any event, the load sensitivity of the CSR algorithm, its simple specification, and its linear response characteristics establish it as an attractive scheduling scheme for interactive systems.

References

[1] P. Brinch Hansen, "An Analysis of Response Ratio Scheduling," Proc. IFIP Congress 71, Ljubljana, August 1971, pp. TA-3:150-154.

[2] K.M. Chandy, J.H. Howard and D.F. Towsley, "Product Form and Local Balance in Queueing Networks," Journal of the ACM 24, 2 (April 1977), 250-263.

[3] E.G. Coffman and L. Kleinrock, "Computer Scheduling Methods and Their Countermeasures," AFIPS Conf. Proc., SJCC 1968, pp. 11-21.

[4] L. Kleinrock, Queueing Systems, Vol. 2: Computer Applications. John Wiley and Sons, New York, 1976.

[5] J.D.C. Little, "A Proof for the Queuing Formula: $L=\lambda W$," Operations Research 9, 3 (May 1961), 383-387.

[6] R.B. Miller, "Response time in man-computer conversational transactions," AFIPS Conf. Proc., FJCC 1968, pp. 267-277.

[7] R.R. Muntz, "Poisson Departure Processes and Queueing Networks," IBM Research Report RC 4145, December 1972.

[8] D.M. Ritchie, Verbal communication on the UNIX scheduling algorithm, September 1977.

[9] M. Ruschitzka and R.S. Fabry, "A Unifying Approach to Scheduling," Communications of the ACM 20, 7 (July 1977), 469-477.

[10] M. Ruschitzka, "Performance evaluation of nonpreemptive response-ratio schedulers," AFIPS Conf. Proc., NCC 1978, pp. 473-481.

[11] M. Ruschitzka, "An Analytical Treatment of Policy Function Schedulers," Operations Research 26, 5 (Sept.-Oct. 1978), 845-863.

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

PRIORITY BATCH PROCESSING FOR UPPER BOUNDED RESPONSE TIMES

U.De Carlini, A.Mazzeo, C.Savy
Centro di Calcolo Elettronico
Facoltà di Ingegneria-Università di Napoli
Naples, Italy

The case when a set of job classes for which time service distributions are provided is herewith investigated with reference to a nonpreenptive priority (HOL) batch scheduling. Server is single and the job allocation discipline into priority queues depends on the class and the expected service time. A requested extreme response time is given to each class depending on the service time and a criterion is provided to choose the number of priority queues to service so that the number of unsatisfied requests may drop to a minimum.

INTRODUCTION

It is well known that batch is the most common among job processing techniques employed by data computing systems. Among several others, such technique is widely used being quite simple to perform and, compared to other procedures, it may provide a better system throughput. This method may be implemented either on mono-or multi-programmed systems. A limitation to traditional batch processing comes from it keeping absolutely in no account users' request of reasonably short response times compared to the time the job takes and/or to the charged tariffs. Therefore priority job scheduling was introduced into batch processing. Scheduling algorithms, at times even quite sophisticated, often allow for an arrangement of jobs into classes and, within each class, take in consideration their expected processing time |1|. A static initial allocation is often followed by a dynamic one which consideres the age and/or the accumulated service time of the job |2|.

Several are the scopes to meet with a priority job scheduling. Among these, the most common ones are |3|:

1)to advantage some of the jobs according to their nature or to the applied tariff |4|;
2)to minimize the system's mean response time;
3)to satisfy as many users possible within reasonable response times, eventually charging diversified tariffs.

Priority scheduling assumes that submitted jobs be distributed along queues. The number of queues and the criteria for which jobs are assigned to them are strictly connected to the pursuing objectives. With reference, for instance, to case (1) where jobs are divided into classes according to their nature, the number of priority queues is assumed to be the same as that of the job classes to serve. In case (2) instead, a single job class exists and there are many queues along which distribute jobs according to the required processing time. It is known indeed from queue theory that mean response time tends to a minimum if jobs are divided among

a number of queues tending to infinity, with queue priority growing as the requested service time decreases (SJF discipline, [5]). If a queueing discipline is to be studied, to meet case (3) scopes, unlike the former ones it is non-obvious as several parameters need be defined and accounted for. Parameters depend from required response times as much as from the computer's workload and utilization factor.

Having defined the requested response times for each job class as a function of the required service time, it is the scope of this paper to show that satisfactory requested response times on one hand depend on the system's workload and utilization factor and, on the other, on the jobs' allocation along queues. Dependency is well noted by means of a schematic model of a computing system. Although being simple, the model produces useful results either when analysing or dimensioning a computing system.

During analysis, assuming jobs have been distibuted among classes and that requested response times and the system's workload and utilization coefficient have been defined, the model makes it possible to analyse various queueing disciplines and see if there is one which meets all required specifications. If none exists, the model gives informations on what part of the load may be processed within specification limits.

With reference to the dimensioning phase, once workload characteristics and a queueing discipline have been given, the model tells what capacity should the system have if jobs are to be serviced within requested response times.

THE PROPOSED MODEL

The computing system is regarded by the model (Fig.1) as a single server which serves n_γ queues ($n_\gamma \geq 1$) into which jobs, to be priority and non preemptively served, are distributed [6]. Jobs are divided into n_c classes. Each job is characterized

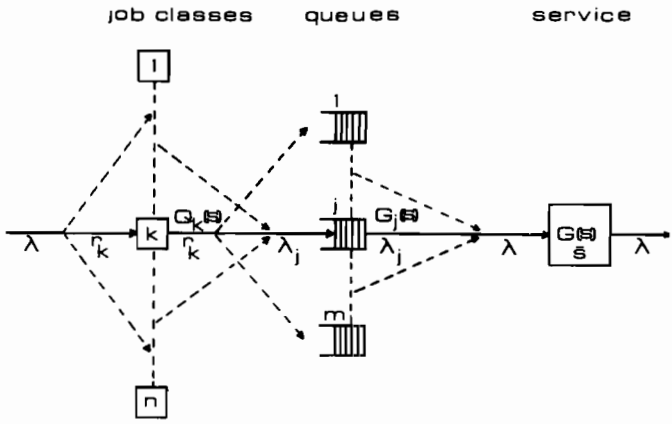


Fig.1 - Model of The Computing System

by a specific required service time s and by a requested maximum response time $t_{rd}(s,k)$ which depends both on the required service time and on the class k ($1 \leq k \leq n_c$) the job belongs to. We assume that a job is *satisfactorily processed* if jobs with service time s belonging to class k show a mean response time $t_q(s,k)$ no greater than $t_{rd}(s,k)$. Mean response time $t_q(s,k)$ depends on:

1) the division of jobs into classes and on the distribution of arrivals to each one of them;
2) the number of queues n_l and on the rule of allocation to each queue;
3) all jobs' service times distribution and on the service times distribution of jobs belonging to each queue.

With reference to (1), we assume that each job is submitted to the system at random so that the probability of a particular job being submitted in a given time period is very low and constant. It is therefore possible to employ, for each class k, Poisson's r_k-rate arrivals distribution. Distribution of all classes arrivals is still Poissonian, its rate λ is given by:

$$\lambda = \sum_{k=1}^{n_l} r_k$$

With reference to (2), we assume that $\forall j \in 1,n_l$ a particular job is given to queue j because of the class it belongs to and of the service time it requires. Once the time function t(k,j) has been defined for each job class k and for each queue j, all jobs requiring a service time s, for which

$$t(k,j-1) \leq s \leq t(k,j) \quad \forall k \in 1,n_c$$

will arrive at queue j. It is so possible to assign jobs with even very different service times to the same queue: service priority is thus defined as a function not only of the required time but also of the class the job belongs to.

Supposing each job's expected service time is independent from the arrival time and calling $Q_k(s)$ class k's expected service times cumulative distribution, it may be shown that queue j's arrivals distribution is still Poissonian with rate

$$\lambda_j = \sum_{k=1}^{n_c} r_k \{Q_k(t(k,j)) - Q_k(t(k,j-1))\}$$

and obviously

$$\lambda = \sum_{j=1}^{n_l} \lambda_j$$

Therefore, being:

- $\bar{s}$ all jobs' mean actual service time,
- ρ the system's utilization factor,

and with reference to queue j:

- $\bar{s}_j$ mean actual service time of all entering jobs,
- $\bar{t}_{wj}$ mean waiting time in the queue,

the following relations may be written [7]:

$$\rho = \lambda \bar{s}$$

$$\bar{t}_{wj} = \frac{\rho \bar{s}^2}{2\bar{s}\left(1 - \sum_{i=1}^{j-1} \lambda_i \bar{s}_i\right)\left(1 - \sum_{i=1}^{j} \lambda_i \bar{s}_i\right)} \qquad (1)$$

As far as point (3) is concerned, the model assumes that the job's actual service time is the same as the required service time. Therefore once the required service times cumulative distribution $Q_k(s)$ and the time function $t(k,j)$ are known, actual service times cumulative distributions $G(s)$ and $G_j(s)$, respectively related to all jobs and to those entering each queue, may be evaluated. Parameters $\bar{s}$ and $\bar{s}_j$ $\forall\, j \in 1,n_q$ are thus known as well as each queue's mean waiting time $\bar{t}_{wj}$ and each job's mean response time

$$\bar{t}_q(k,s) = \bar{t}_{wj} + s$$

Expressing $\bar{t}_{wj}$ as a function of k, $\bar{s}$, ρ, n_q, only, we may write

$$\bar{t}_q(k,s) = f(\bar{s},\rho,n_q,k,s)$$

It is therefore possible to evaluate, as a function of the four parameters k, $\bar{s}$, ρ, n_q, those values of s for which the equation

$$t_{rd}(k,s) - \bar{t}_q(k,s) = g(\bar{s},\rho,n_q,k,s) = 0 \qquad (2)$$

is satisfied. Plotting such values into the space s, $\bar{s}$, ρ, n_q, k we have boundary surfaces which separate regions where jobs are processed satisfactorily by the system ($g > 0$) from those where they are not ($g < 0$). Obviously, if three of the four parameters are given it is possible to study how the limit between satisfactory responses varies with respect to the fourth.

If, for instance, we suppose ρ as the fourth parameter, plotting equation (2) onto the plane ρ, s, we have a diagram as the one shown in Fig.2 which shows how the boundary between satisfactory and unsatisfactory responses varies with respect to ρ. Referring to the figure, considering for instance $\rho=\rho'$, on the s axis we get the two values s_1, s_2. Satisfied jobs are therefore those, belonging to the class the diagram is referred to, whose response times are included between s_1 and s_2, while jobs whose service times are either smaller than s_1 or greater than s_2 will not be satisfied by the system.

USE OF THE MODEL

A few illustration are given hereafter to show how the model may be used either during a system's analysis stage or its dimensioning. Assumption is made that in each example requested response times vary linearly with required service times according to coefficients depending from the class k the job belongs to:

$$t_{rd}(k,s) = C_k + B_k s$$

Coefficient C_k makes due allowance for that part of the response time which may at any rate be considered as being independent from the service time and propor-

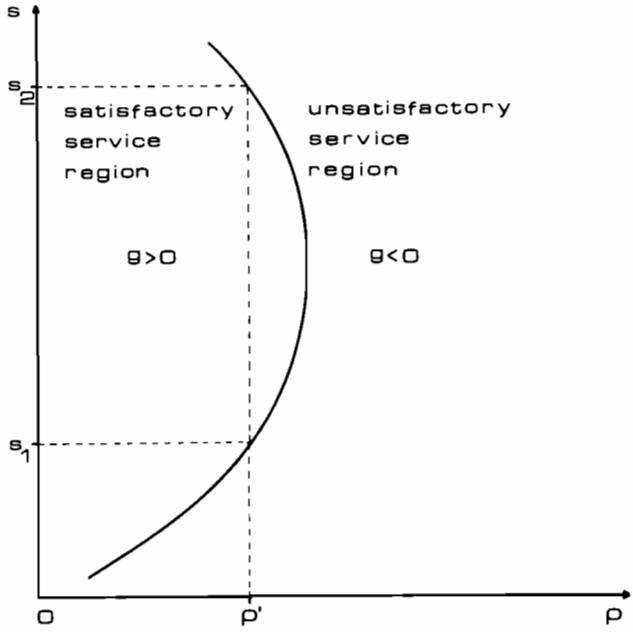


Fig. 2 - Satisfactory Service Region as Function of Utilization Factor

tional to the system's mean service time s. The requested response time is then given by the relation:

$$t_{rd}(k,s) = A_k \bar{s} + B_k s \qquad (3)$$

Obviously other dependence laws, as the exponential one for instance, may be stated between requested response time and service time. At any rate, choice of the best functional relation depends on several elements whose analysis goes beyond the scope of this paper.

As first illustration, let there be a batch processing system operating at an University site. Scientific, administrative and students' jobs are submitted to the system. Obviously, there will be no firmly defined limits imposed on submitted jobs response times, but we may logically suppose that students and other users who present scientific jobs expect response times being *reasonable*, or, at any rate, proportional to their jobs' service time. We may therefore suppose that the only class the system schedules on a priority base according to the required computing time is made of scientific and students' jobs only. Requested response time is supposed given by (3). Administrative jobs will instead be serviced as background work.

Supposing we are analysing the system's performances, that the cumulative distribution of submitted jobs' service times is known and that parameters A and B of (3) are given, we wish to see, for different queue allocation rules, how requested response times compliance depends on the system's utilization factor, as computed for scientific and students' jobs. This investigation makes it possible either to find the most appropriate queueing rule or to evaluate the maximum scientific and students' workload serviceable within response times' restraints.

As this illustration makes reference to no particular computing centre, we shall distinguish between two different theoretical service time distributions: uniform and exponential. With reference to the queueing rules to be examined, time t(j)

$$t(j) = (2\bar{s}/n_l)\cdot j \qquad j \neq n_l$$

is given to each queue j, while queue n_l is given the maximum service time according to the examined distribution. Jobs whose service times are given by

$$t(j-1) \leq s \leq t(j) \qquad j \neq n_l \tag{4}$$

will be allocated to queue j while those whose service times are greater than $t(n_l-1)$ well be allocated to queue n_l.

Case A: uniform distribution

Under such assumptions service times are included in the range $0 \leq s \leq 2\bar{s}$, their cumulative distribution being

$$G(s) = s/2\bar{s}$$

therefore

$$\bar{s}^2 = (4/3)\cdot\bar{s}^2$$

Queue j's mean arrival rate λ_j and the mean service time $\bar{s}_j$ of all jobs entering it are given by

$$\lambda_j = \lambda/n_l$$

$$\bar{s}_j = (j-\tfrac{1}{2})\cdot 2\bar{s}/n_l$$

Replacing these values into (1), we get queue j's mean waiting time

$$\bar{t}_{wj}(\bar{s},\rho,n_l) = \frac{(4/3)\bar{s}\rho}{2(1-\rho(j-1)^2/n_l^2)\cdot(1-\rho j^2/n_l^2)} \tag{5}$$

It is now possible to write (2) as

$$A\bar{s} + s(B-1) - \bar{t}_{wj} = 0 \tag{6}$$

being $\bar{t}_{wj}$ given by (5) and j fixed so that $\forall\, s \in 0,2\bar{s}$ relation (4) is be satisfied. Equation (5) stands for a finite number of queues. In order that the queueing rule may be examined as n_l grows, we shall consider the case where an infinite number

of queues exist. A strictly prioritary discipline may be employed and shorter jobs will be given a higher priority (SJF). Under such assumption, jobs whose service times are within the range s, s+ds arrive at the queue associated to service time s. We may write for a generic queue:

$$\lambda_s = s\lambda/2\bar{s}$$

$$\bar{s}_s = s^2\lambda/4\bar{s}$$

$$\bar{t}_{ws} = \bar{s}^2/3(1-\lambda\bar{s})$$

The plot of the equation (6) on the ρ, s plane when $n_l = 1$ and $n_l = \infty$, supposing $\bar{s} = 60$ sec, A = 1 and B = 2, is given in Fig.3.

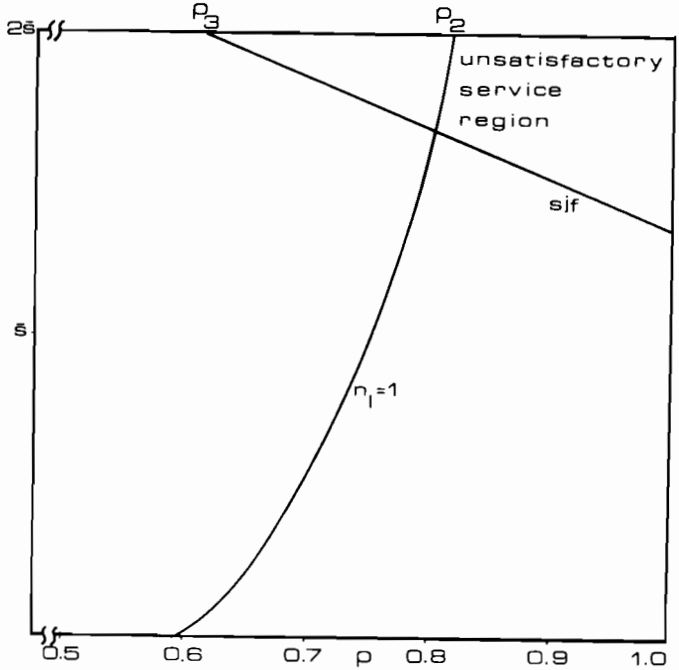


Fig.3 - Case A: Plot of Equation (6) on the ρ,s plane (n_l=1,∞)

Quite general considerations may be made on jobs' priority processing using these diagrams. We may particularly notice that in FIFO (n_l=1) servicing there is a limit utilization factor value ρ_1 (ρ_1= 0.6 in Fig.3) corrisponding to which any job's mean response time are lower than the given limit. For values of ρ in the range ρ_1, ρ_2 (ρ_2= 0.8 in Fig.3) shorter service time jobs will not be processed within the required response times while satisfactory service will be given to jobs with greater service time. Finally, if $\rho>\rho_2$ the FIFO discipline cannot process jobs within requested mean response times.

On the other hand, a strictly priority service discipline (SJF), promotes shorter service time jobs. It may in fact be seen from Fig.3 that if $\rho > \rho_3$ ($\rho_3 \simeq 0.6$) this discipline will not process jobs which require a higher service time within mean requested response times.

Comparing both cases, we notice that there is a part of the ρ, s plane where both disciplines may be satisfactorily employed, another where neither may be employed with satisfaction, and other still where one of the two only is satisfactory. Actually, the SJF discipline was considered only as an extreme case of reference, as use of a discipline with priority strictly bound to service times is not practical since service times cannot be exactly defined in advance. Analysis will thus have to be restricted to a relatively small number of queues: Fig.4 shows cases for which $n_l = 2, 4, 6, 8$.

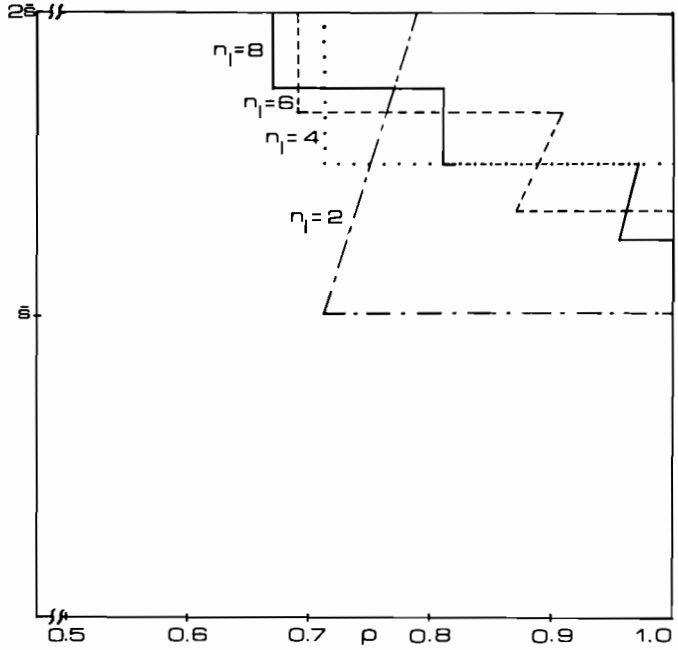


Fig.4 - Case A: Plot of Equation (6) on the ρ,s plane ($n_l = 2, 4, 6, 8$)

We may particularly notice that, within the four given values, when $n_l = 4$ the region of unsatisfactory service is already sufficiently reduced with respect to $n_l = 1$ and close to the SJF case.

Examining the diagrams, it may be noticed that with high values of ρ and with growing n_l the region of unsatisfactory service is referred to jobs arriving at the last queue and progressively extends to those arriving at immediately higher priority queues. Consequently, if ρ is high, the number of satisfactory service

queues may not be generally defined as the choice depends upon ρ's value. Fixing ρ, the choice could be made employing such a number n_l of queues that only jobs assigned to the lowest priority queue will be unsatisfactorily serviced, while those assigned to the immediately higher priority queue will be at the limit of satisfactory service. Being for instance ρ=0.8, n_l= 6 could be chosen.

All plotted diagrams, repeated at different values of $\bar{s}$, have shown to be practically independent from such parameter and will therefore hold whatever the system's mean service time.

Case B: exponential distribution

In such case, service times fall within the range $0 \leq s < \infty$ their cumulative distribution being given by

$$G(s) = 1 - e^{-s/\bar{s}}$$

from which

$$\bar{s^2} = 2\bar{s}^2$$

By operating as in the previous case, it is possible to solve equation (2) and evaluate each queue's mean waiting time. Equation (2) is plotted on the ρ,s plane (Fig.5), still being $\bar{s}$=60 sec, A=1, B=2, n_l= 1, 2, 4, 6 within the range $0 \leq s \leq 2\bar{s}$ as in the case of uniform distribution. It may be noted that these diagrams are qualitatively similar to the previous case ones, with a slight increase of the unsatisfactory service area. It may also be noticed that the smallest boundary ρ, for which jobs arriving at the lowest priority line receive satisfactory service, depends on the arrival at that queue of jobs whose service times are greater than $2\bar{s}$. This jobs are of course not present if a uniform time service distribution is assumed. Considerations made in the case of uniform distribution, will still hold. All given diagrams, plotted also for other values of $\bar{s}$, have shown to be practically independent from such parameter and will hold whatever the system's mean service time.

Case C: privileged jobs

The assumption n_c= 1 (all jobs placed into a single class) is often unrealistic when priority batch servicing, because highly urgent jobs are present. Namely, these jobs are those directly presented by the head of the centre because of management purposes, and those eventually charged differently. Therefore, submitted jobs quite often have to be divided into two classes at least, the privileged one being serviced with maximum priority, independently from the required service time.

It may be interesting to examine a job queueing rule for which jobs belonging to the privileged class (c_1) will enter the maximum priority queue, while jobs belonging to the other class (c_2) will enter the other queues according to the previously described queueing rule. It is therefore possible to evaluate within which range may the privileged workload vary without the remaining workload's response times degrading untolerably. Such policy was examined assuming same mean value $\bar{s}$ uniform service times distributions for both classes, and arrival rates

$r_1 = (1 - p)\lambda$

$r_2 = p\lambda$

for classes c_1 and c_2 respectively.

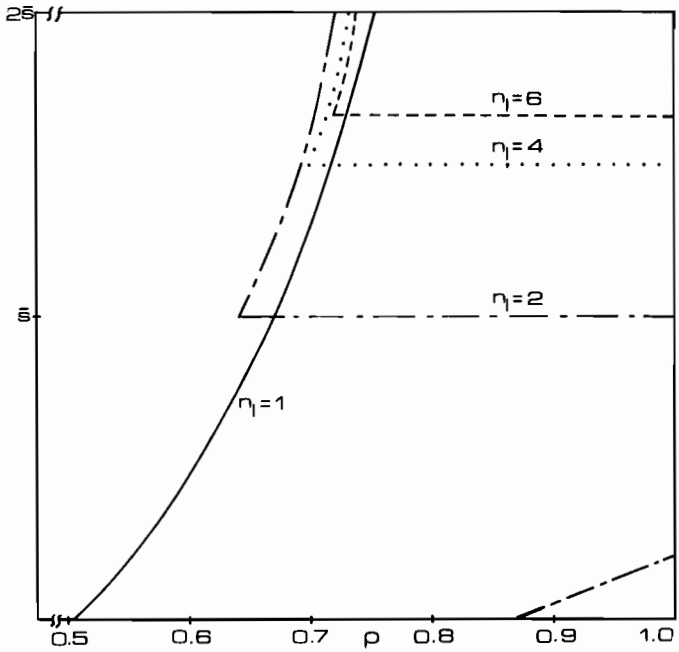


Fig.5 - Case B: Plot of Equation (2) on the ρ,s plane (n_1= 1, 2, 4, 6)

Equation (2) is plotted on the ρ, s plane for class c_2 (Fig.6), when p = 0.8, $\bar{s}$ = 60 sec, A = 1, B = 2, n_1= 1, 2, 4, 6.

CONCLUSION

A model is proposed in this paper for the investigation of priority scheduling techniques applicable to batch processing systems. Having chosen a set of possible scheduling rules and defined the function *requested response time/required service time* the model shows which rule is the most appropriate, suggesting how to share jobs among priority queues.

The model is a simplified one, as it characterizes the computing system and its workload simply by:
- the system's utilization factor,
- the time service distribution of jobs for which an exponential arrival distribution is assumed,

- the mean service time.

All these parameters may be experimentally evaluated from a real system.

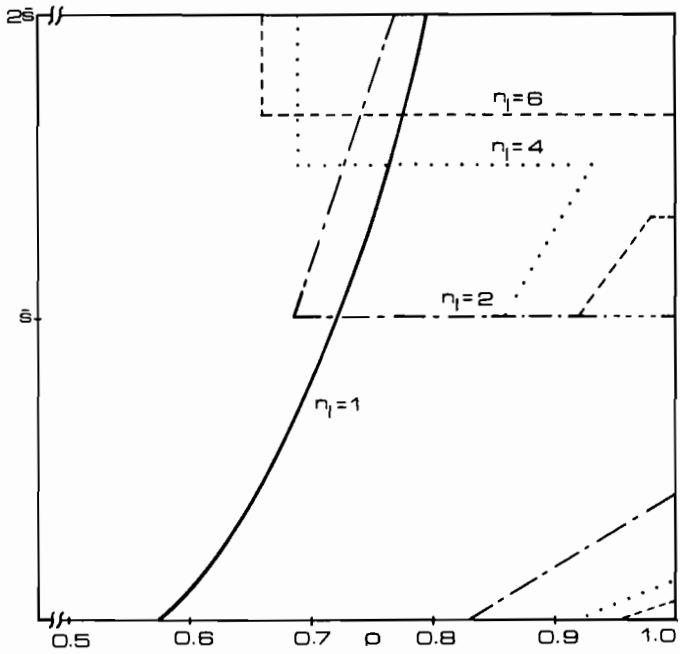


Fig.6 - Case C: Plot of Equation (2) on the ρ, s plane (n_l= 1, 2, 4, 6)

Considering the theorical service time distributions, the uniform and exponential ones, it was shown that the model's behaviour is qualitatively independent from the chosen distribution and that results are practically independent from the adopted mean service time. Within our model's limits, quite generally sound results were found and given in a practically adimensional format.

With reference to monoprogrammed systems, the model matches the real system quite closely and will provide reliable results. On the other hand, in case of multiprogrammed systems, our model shows obvious limitations, among which the following should be mentioned:

- actual service times supposed independent from the system's state and depending on the job only,
- strictly sequential job servicing,
- static scheduling rule.

While stating the model's limitations, we wish to add that a wider definition of the same requires that many parameters, not always readily evaluated in practice, be introduced. On the other hand, results from using the model with multiprogrammed systems are more meaningful if correction factors are taken into account.

Namely, when defining the system's utilization factor, a gain should be considered to come from multiprogramming, which should quantify how faster the workload is processed with such service. Furthermore, as the service time given by the system to each job is conventional, it will not represent the actual time between beginning and ending of service anymore: therefore when evaluating the mean response time, we should replace the service time with a different time which should roughly be proportional to the service time according to a parameter which depends on the system and its workload.

Application of the model to a particular multiprogrammed system is presently being studied |8|. The arrival and service distributions are taken from the system's log as well as each job's response time, service time and conventional service time. Scope of this investigation is to find a particular queueing rule for a practical problem, so that the number of jobs over the required service time is minimized. A further analysis will be made on privileged jobs' effects on response times.

REFERENCES

|1| K.C. Sevcik: Scheduling for Minimum Total Loss Using Service Time Distribution, Journal of ACM, vol.21, n°1, (1974).

|2| H.W. Lynch, J.B. Page: The OS/VS2 Release System Resources Manager, IBM System Journal, vol.13, n°4, (1974).

|3| D.N. Streeter: Cost-Benefit Evaluation of Scientific Computing Services, IBM System Journal, vol.11, n°3, (1972).

|4| J.R. Holt: The Automatic Control System at Boeing Computer Services, Proc. of ECOMA-5, London, April 1978.

|5| T.E. Phipps: Machine Repair as a Priority Waiting-Line Problem, Operations Research, vol.4, (1956).

|6| L. Kleinrock: Queueing Systems, Wiley Interscience, New York, (1976)

|7| T.L. Saaty: Elements of Queueing Theory, McGraw-Hill, New York, (1961).

|8| U. De Carlini, A. Mazzeo, C. Savy: Jobs' Optimal Queueing Discipline for the Computing System of University of Neaples. To be published.

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

WAITING-TIME DISTRIBUTIONS FOR DEADLINE-ORIENTED SERVING *

B. Walke, W. Rosenbohm
AEG-TELEFUNKEN, Research Institute Ulm,
West Germany

An infinite queue single server model is considered where requests of type i, $1 \leq i \leq N$ arrive from independent Poisson streams and demand service according to arbitrary d.f.'s which may be different for different types of requests. Associated with each type-i request is an urgency number v_i which, together with the request's time of arrival, defines a deadline for beginning its service. Preemption of a request in service is not permitted. This relative urgency (RU) discipline has at its two limiting cases the FCFS and head of the line (HOL) disciplines. In [1] the mean waiting time is computed approximately and close bounds are derived there. Here we derive close approximations for the tails of the waiting time d.f.'s and compare them to those of the HOL and FCFS disciplines. Moreover, some properties of the preemptive version of the RU discipline are presented.

1. INTRODUCTION AND DEFINITION OF MODEL

In real-time computer-control systems it is important that incoming service requests from a running process be completed in time, i.e. within a given time limit. In this paper we will consider the case of requests arriving at random (not predictable) times. Given this randomness it is impossible to guarantee that all finite deadlines for beginning or ending these requested services can be met. The best that can be done is to guarantee a high probability for meeting such deadlines.

Let us consider a computer model with unlimited waiting space for incoming requests which may possess various properties (to be denoted in the following as "types"). Further suppose these type-i requests ($1 \leq i \leq N$) arrive at a rate λ_i from a Poisson process, have an arbitrary service-time b_i according to a distribution function $F_i(t) = P_i(b_i \leq t)$ with a finite second moment, and that each request has a deadline t_i for beginning based on its arrival time T_i and its urgency v_i. Without any loss of generality we order the types i of requests such that

$$0 \leq v_1 < v_2 < \ldots < v_i < \ldots < v_N \tag{1.1}$$

where N is the total number of types and v_i is the urgency of a type-i request. The smaller the value of v_i, the higher the urgency.

Figure 1 shows a model of our system. We will be concerned almost exclusively with the case of requests which, once servicing has begun, may not be interrupted (non-preemptive priority). After a request has been serviced, the request among those waiting to be processed which has the highest dynamic priority $q_i(t)$ is chosen for servicing. The dynamic priority at time t is given by

$$q_i(t) = v_i - t + T_i \ , \tag{1.2}$$

where T_i is the arrival time and v_i the urgency. If we define the waiting time

* This work was partly supported by the 2nd EDP Program of the Federal Government of West Germany.

$w_i(t)$ at time t as

$$w_i(t) = t - T_i \tag{1.3}$$

then the current priority is

$$q_i(t) = V_i - w_i(t) \tag{1.4}$$

and depends linearly only on the current waiting time. As usual small values of $q_i(t)$ indicate a high priority. It is interesting to note that in this model the priorities of all outstanding requests, regardless of type, grow uniformally. This is illustrated in [1] with an example. Requests of the same type are processed in the order they arrived (first come first serve FCFS). Since requests are serviced according to individual urgencies and arrival times, we speak of a relative-urgency (RU) discipline.

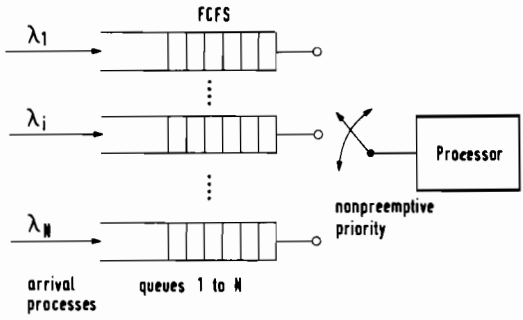


Fig. 1: Model of a real-time computer-control system λ_i = arrival rate.

In addition to the non-preemptive discipline (RU-NONPRE) we will consider a few examples of a preemptive RU discipline (RU-PRE). In the RU-PRE discipline a type-i request may be interrupted (preempted) by a type-j request if $w_i < V_i$ and $w_j = V_j$. The priority $q_i(t)$ of a request remains unchanged once its servicing has begun. But if a preemption occurs, the priority of the interrupted request again follows Eq.(1.4) with $w_i(t)$ being the total time such a request remains in the system without being serviced. The urgency V_i represents the time w_i a request is willing to wait from its arrival T_i to its deadline t_i. For both the non-preemptive and preemptive versions of the RU discipline a request's deadline may be defined to be reached if $w_i(t)$ equals V_i. If the actual waiting time w_i of a type-i request is less than its voluntary waiting time V_i, while another type-j request has already waited its voluntary time V_j, then the type-j request preempts the type-i request. The case i = j is impossible. Requests, whose servicing just barely began on time or which missed their deadlines may not be interrupted. Summarizing, it can be said that in the RU-PRE discipline only requests whose servicing began ahead of schedule may be interrupted and only by requests whose deadline has been reached.

The following abbreviations are used throughout:

w_i, b_i	Random variables: waiting and service time of a type-i request
d.f., d.f.'s, w.t.d.f.	distribution function, d. functions, waiting time d.f.
$W_i(t) = P(w_i \le t)$	w.t.d.f. of a type-i request
$W_i^{(r)}$, $W_i^{(1)} = W_i$	r-th moment of $W_i(t)$
$\lambda_{\le N} = \sum_{j=1}^{N} \lambda_j$	total arrival rate of all requests together
$W_{\le N}(t) = \lambda_{\le N}^{-1} \sum_{i=1}^{N} \lambda_i W_i(t)$	common w.t.d.f.

$F_i(t) = P(b_i \le t)$ service time d.f.

$\beta_i^{(r)}$, $\beta_i^{(1)} = \beta_i$ r-th moment of service time d.f.

$\rho_i = \lambda_i \beta_i$ offered traffic of type-i requests

$\rho_{\le N} = \sum_{j=1}^{N} \rho_j$ total offered traffic

$F_{\le N}(t) = \lambda_{\le N}^{-1} \sum_{i=1}^{N} \lambda_i F_i(t)$ common service time d.f.

$\beta_{\le N}^{(r)} = \lambda_{\le N}^{-1} \sum_{i=1}^{N} \lambda_i \beta_i^{(r)}$ r-th moment of $F_{\le N}(t)$

$C_{\le N}^2 = \beta_{\le}^{(2)} / \beta_{\le N}^{(2)} - 1$ squared coefficient of variance of $F_{\le N}(t)$

$\bar{\nu} = \rho_{\le N}^{-1} \sum_{i=1}^{N} \rho_i \nu_i$ mean weighted urgencies

$Var(\nu) = \sum_{i=1}^{N} \frac{\rho_i}{\rho_{\le N}} \nu^2 - \bar{\nu}^2$ variance of the urgency numbers.

The RU discipline uses dynamic priorities based on waiting times. The problem described at the beginning of this paper nowadays is usually treated with a service discipline which recognizes only static priorities. Arriving requests are assigned a static priority i according to their type i. Among the waiting requests the one with the highest priority (lowest i) which has waited the longest of all requests of the same type i (FCFS) is processed first. Here, too, the model in Figure 1 applies, except that there is no urgency ν_i.

If one is interested in the probability of missing deadlines, then one should know the waiting time d.f. (distribution function) $W_i(t) = P(w_i \le t)$. From this the probability that the waiting time w_i is not greater than t can be obtained. The properties of static and dynamic priorities with regard to the divergence of the resulting waiting times can be discussed by comparing their waiting time d.f.'s. Such a comparison shows how much the probabilities for meeting given deadlines $t = \nu_i$ differ between static and dynamic priorities. Since this comparison depends on the type of request considered, the sum of the difference $[W_{istat}(\nu_i) - W_{idyn}(\nu_i)]$ between static and dynamic priorities weighted by the probability of occurance $\lambda_i/\lambda_{\le N}$ of type-i requests (which is independent of discipline) is useful. In this manner one obtains a formal figure of comparison V:

$$V = \lambda_{\le N}^{-1} \sum_{i=1}^{N} \lambda_i [W_{istat}(\nu_i) - W_{idyn}(\nu_i)] \quad . \tag{1.5}$$

From this we see that:

if $V < 0$, then a relativ urgency discipline is better, and
if $V > 0$, then a discipline with static priorities is better. (1.6)

From the definition of V we have $-1 \le V \le 1$.

We will consider the model in Figure 1 only in a state of equilibrium, which occurs only when the total offered traffic $\rho_{\le N} < 1$.

In disciplines with static priorities, a stationary equilibrium is achieved for requests with high priorities, sometimes even when $\rho_{\le N} > 1$. However, we will not consider this case here.

2. WAITING TIME D.F.'s FOR VARIOUS DISCIPLINES, EXAMPLES

Let us assume N different types of requests, each with its arrival rate λ_i, urgency V_i, and service time d.f. $F_i(t)$. Since the waiting time d.f. cannot be calculated, we will consider four examples where were simulated.

Example 1: Modell M/M/1, N = 4,

$\rho_i = \rho_{\leq N} / N$, $\beta_{\leq N} = 2.386$, $C_{\leq N} = 1.52$,
$\bar{V} = 20$, $Var(V) = 125$.

All requests have a negative exponential distributed service time with mean β_i.

i	1	2	3	4
β_i	1	3	5	7
V_i	5	15	25	35

Example 2: Modell M/D/1, N = 4,

$\rho_i = \rho_{\leq N} / N$, $\beta_{\leq N} = 2.386$, $C_{\leq N} = 0.81$,
$\bar{V} = 30$, $Var(V) = 125$.

All requests have a constant service time.

i	1	2	3	4
β_i	1	3	5	7
V_i	15	25	35	45

Example 3: Modell M/G/1, N = 4,

$\rho_i = \rho_{\leq N} / N$, $\beta_{\leq N} = 1.86$, $C_{\leq N} = 2.96$,
$\bar{V} = 20.5$, $Var(V) = 94.25$.

Service time d.f.'s of type-1 and type-4 requests are hyperexponential of order 2. Requests of type 2 and 3 have constant and negative exponential distributed service times, resp.

i	1	2	3	4
β_i	2.45	1	3	2.45
d.f.	H_2	D	M	H_2
V_i	9	14	25	34

Example 4: Corresponds to example 2 but with inverse order of the mean service time β_i with regard to the type-i.

i	1	2	3	4
β_i	7	5	3	1
V_i	15	25	35	45

The examples given here differ in part markedly in the service time d.f.'s assumed for the individual types of requests and in the coefficients of variance $C_{\leq N}$ of the common service time d.f.

From an earlier investigation (cf. [1] and [4]) it is known that the smaller the variance of the urgency numbers is, the less the mean waiting times W_i of the RU-NONPRE discipline differ from the mean waiting time

$$W_{FCFS} = 1/2\ \lambda_{\leq N}\ \beta^{(2)}_{\leq N} / (1 - \rho_{\leq N}) \tag{2.1}$$

in the first come first serve discipline. (For $\lambda_{\leq N}$, $\beta^{(2)}_{\leq N}$ see the list of abbreviations). On the other hand if the variance $Var(V)$ is very large, the RU-NONPRE discipline produces mean waiting times approximating those of the discipline with static non-preemptive priorities

$$W_{iNONPRE} = \frac{\lambda_{\leq N}\ \beta^{(2)}_{\leq N}}{2(1-\rho_{\leq i})(1-\rho_{\leq i-1})} \quad . \tag{2.2}$$

Laplace-Stieltjes transforms of the waiting time d.f.'s for these extreme cases are known and their first two moments can easily be calculated.

2.1. Waiting time d.f.'s of the RU-NONPRE discipline

Let us start with the results obtained: The simulation shows that the waiting time d.f.'s when plotted semilogarithmically, asymptotically approach parallel straight lines regardless of the example and the individual parameters. Let $p(w_i)$ be the

equilibrium probability that a type-i request must wait no longer than w_i units of time for service. For $0 \leq f \leq 1$, define

$$Q_i(f) = \inf\{w_i \mid p(w_i) \geq f\} \quad . \tag{2.3}$$

(This is the waiting time corresponding to fractile f in the cummulative waiting-time d.f. of type-i requests. $Q_i(f)$ usually is called the f-quantile.)
Then the results of the simulation of all four examples seem to indicate that for every M/G/1/RU-NONPRE model we have

$$\lim_{f \to 1} [Q_i(f) - Q_j(f)] = v_i - v_j \quad . \tag{2.4}$$

As the probability f approaches 1, the distance between the w.t.d.f.'s for type-i and type-j requests asymptotically approaches the difference in their urgencies v_i and v_j within the accuracy obtainable by simulation. For $f < 1$, this distance is smaller than the difference $v_i - v_j$. Figures 2 through 5 show the results of simulation for example 1 through 4. In each case the total offered traffic is a parameter. For the results shown 1.2 million requests were processed by the simulation model. This large number of requests was necessary in order to insure enough samples of rare large waiting times for reasonably reliable results.

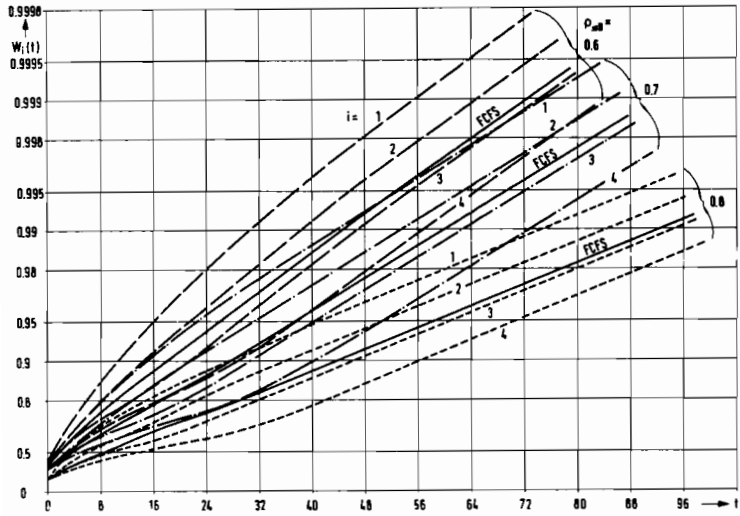


Fig. 2: Waiting-time d.f.'s of the model M/M/1/RU-NONPRE, cf. example 1. The type i and total offered traffic $\rho_{\leq N}$ are parameters. The solid and dashed curves result from the FCFS and RU disciplines, respectively.

Nevertheless, the results shown in Figures 2 through 5 had to be corrected in the tail of the waiting time d.f. (i.e. for large t) by extrapolation. Because of the great rarity of extremely large waiting times in one time-limited simulation experiment where the stationary state could not be completely reached, these large waiting times are underrepresented. Figure 6 shows the true results of example 2 (cf. Fig. 3). The "turning up" of the curves in this figure for large waiting times t is due to the limited number of samples. Experimentally it can be shown that as the number of requests processed per simulation increases, this "turning up" moves to ever greater waiting times. For a given waiting time t, the deviation from the straight line constructed by extrapolation from smaller waiting times becomes less and less.

By comparing the results of the four examples, it can be seen that the waiting time d.f.'s run parallel from about $f = w_i(t) = 0.98$ on when plotted semilogarithmically.

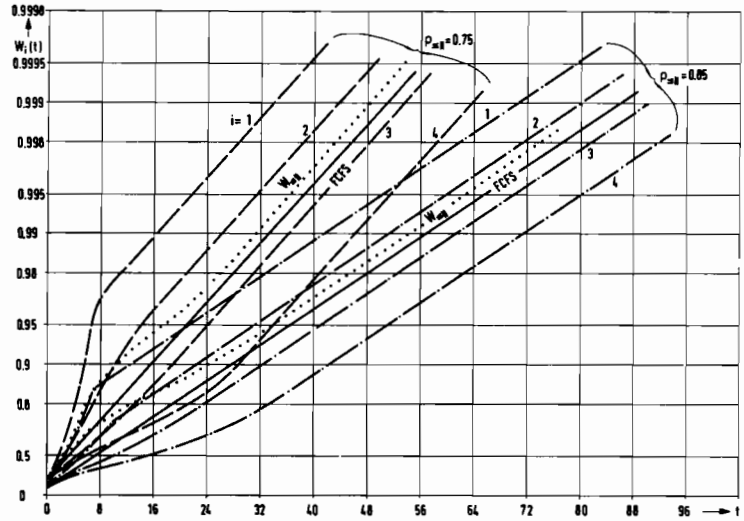


Fig. 3: Waiting-time d.f.'s of the model M/D/1/RU-NONPRE, cf. example 2. The type i and total offered traffic $\rho_{\leq N}$ are parameters. The solid and dashed curves result from the FCFS and RU disciplines, respectively.

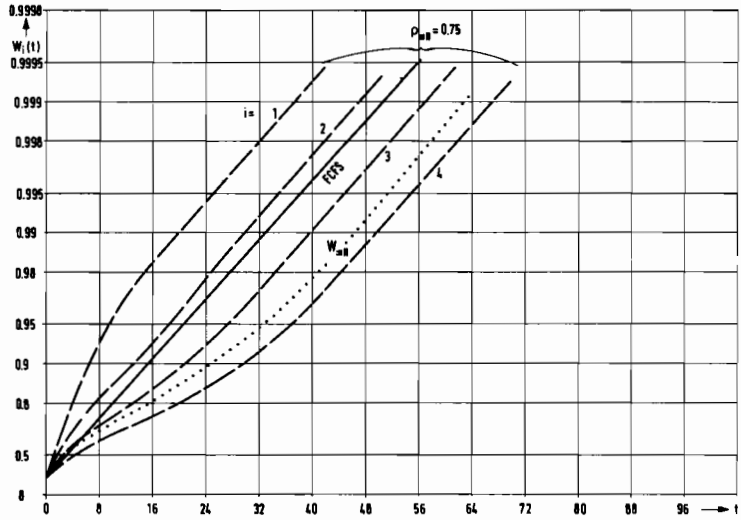


Fig. 4: Waiting-time d.f.'s of the model M/D/1/RU-NONPRE, cf. example 4. The type i and total offered traffic $\rho_{\leq N}$ are parameters. The solid and dashed curves result from the FCFS and RU disciplines, respectively.

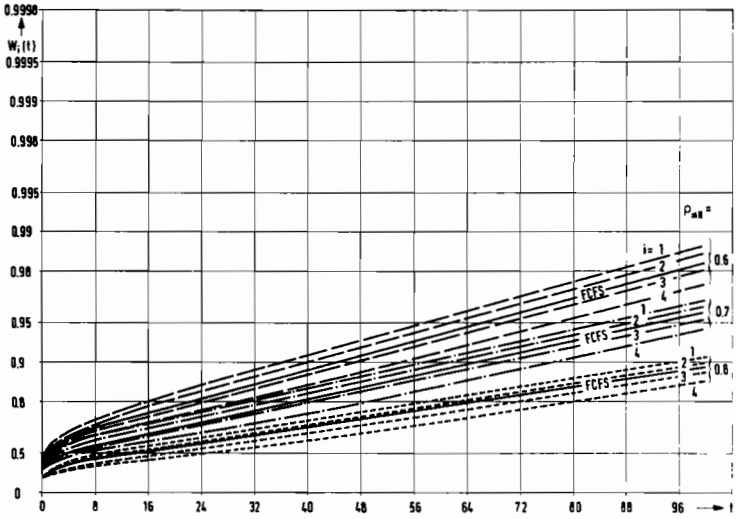


Fig. 5: Waiting-time d.f.'s of the model M/G/1/RU-NONPRE, cf. example 3. The type i and total offered traffic $\rho_{<N}$ are parameters. The solid and dashed curves result from the FCFS and RU disciplines, respectively.

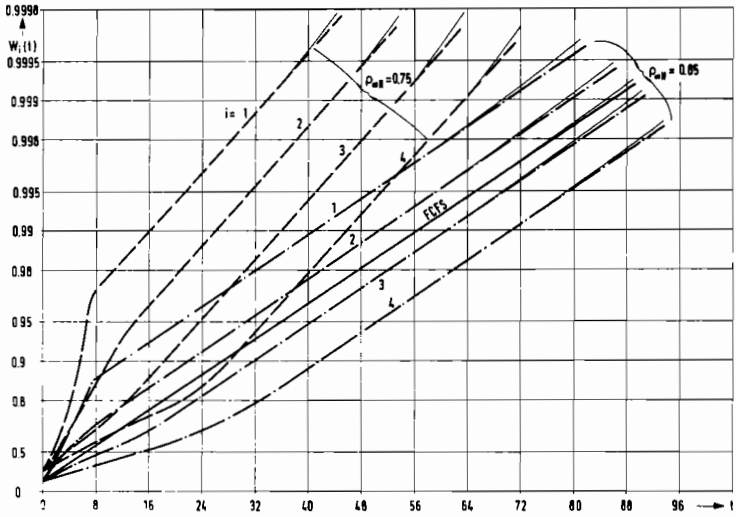


Fig. 6: Extrapolated results and true simulation results for the example 2, cf. Fig. 3.

Very large waiting times of different type requests differ by the difference in their urgencies. Smaller waiting times differ by less. The property formulated in Eq.(2.4) implies that the probability $P(w_i \leq t)$ of meeting deadlines with very large waiting times w_i is practically independent of the type of request, i.e. all requests are handled with equal fairness. If the condition

$$\min_i \{\nu_i\} \gg \max_i \{\beta_i\} , \qquad (1 \leq i \leq N) , \tag{2.5}$$

is fullfilled, then the probabilities of meeting the deadlines $P(w_i \leq \nu_i)$ are about the same, regardless of type. Only example 1 (c.f. Fig. 2) does not satisfy this condition. Here $\nu_1 < \beta_4$ and hence on the average all type-1 requests miss their deadline if they arrive while a type-4 request is being processed. The probability that a type-4 request is being processed is $\rho_{\leq N} / N$ and indeed the results of the simulation show that the difference in probability of meeting their respective deadlines $P(w_1 \leq \nu_1) - P(w_4 \leq \nu_4) \geq \rho_{\leq N} / 4$. This example shows that there is not sense in assigning deadlines without taking the condition in Eq.(2.5) into account. Such cases can be handled better by the RU-PRE discipline to be discussed later.

2.1.1. Previously known results

Back in 1962 Jackson [2] considered a model with discrete time and a non-preemptive RU discipline. The service times of all requests was taken from one geometric distribution, and the requests themselves were taken from a Bernoulli arrival process. The individual urgency ν_i is computed for every arriving request according to a given distribution. Model M1, which corresponds to this discrete-time model but is time continuous, would have a Poisson arrival process and negative exponentially distributed service times which all originate from a common d.f. Model M1 differs from this model in that different-type requests may have different and arbitrary service-time d.f.'s. Therefore, our model is more general.

In [2] two relations are derived for the discrete time model

$$\lim_{f \to 1} [Q_i(f) - Q_j(f)] = \nu_i - \nu_j \qquad \text{(cf. Eq. (2.4))}$$

and

$$\lim_{f \to 1} [Q_i(f) - Q(f)] = \nu_i - \bar{\nu}' . \tag{2.6}$$

$\bar{\nu}'$ is defined by an expression which for model M1 may be approximated by the mean weighted urgencies $\bar{\nu}$, cf. list of abbreviations. In Eq.(2.6) $Q(f)$ is the f-quantile of the waiting-time d.f. of a single-queue FCFS model, cf. Eq.(2.3). In Jackson's model and in the corresponding model M1, not only is Eq.(2.4) fulfilled, but also the f-quantiles of type-i requests in the RU-discipline differ from the f-quantile of the FCFS model by the difference $\nu_i - \bar{\nu}'$.

2.1.2. Waiting time d.f.'s of models in the RU and FCFS disciplines

Our simulation results of a FCFS model, which can easily be constructed from a model simulating the RU discipline by setting $\nu_i = r$, $(r \geq 0$, real) support the suspicion that Eq.(2.6) seems to be generally applicable, regardless of the service time d.f.'s assumed. From Fig.'s 2 through 5 it can be observed that $\bar{\nu}'$ deviates from $\bar{\nu}$ by no more than 9 %. These deviations are considerably larger than those observed by Jackson for his model. This is due to the fact that unlike Jackson's model the mean service times β_i in our examples are not equal, but different.

The difference $(\bar{\nu} - \bar{\nu}')$ is, depending on the example, greater or smaller than zero. The reason for this can be plausibly explained as follows:

We assume the model in Fig. 1 with non-preemptive static priorities and argue using the results for the common mean waiting time (c.m.w.t)

$$W_{\leq N} = \lambda_{\leq N}^{-1} \sum_{i=1}^{N} \lambda_i W_{i\,\mathrm{NONPRE}} . \tag{2.7}$$

The c.m.w.t. results from weighting and summing over the individual mean waiting times of priority levels i. The c.m.w.t. $W_{\leq N}$ equals the m.w.t. W_{FCFS}, cf. Eq.(2.1), only if all mean service times β_i are equal. This is not the case in example 1 through 4.

From [3] it is known that $W_{\leq N}$ can be minimized in a static priority model by satisfying the condition

$$\beta_1 \leq \beta_2 \leq \cdots \leq \beta_i \leq \cdots \leq \beta_N \quad , \tag{2.8}$$

where 1 is the highest priority. The kind of service time d.f.'s themselves do not matter; only Eq.(2.8) must be satisfied. Since $W_{\leq N}$ and W_{FCFS} are the means of the corresponding waiting time d.f.'s $P(w_{\leq N} \leq t)$ and $P(w_{FCFS} \leq t)$, respectively, then for the case of unequal β_i's, if Eq.(2.8) is satisfied, the w.t.d.f. $P(w_{\leq N} \leq t)$ must be better in some sense than $P(w_{FCFS} \leq t)$.

The opposite should be expected if the worst possible priority assignment is chosen, namely $\beta_i \geq \beta_{i+1}$. The w.t.d.f. $P(w_{\leq N} \leq t)$ is simply a weighted combination of the w.t.d.f.'s $P(w_{iNONPRE} \leq t)$ of the static priority model using

$$w_{\leq N} = \lambda_{\leq N}^{-1} \sum_{i=1}^{N} \lambda_i \, w_{iNONPRE} \quad .$$

It follows that depending on how well or badly the relation in Eq.(2.8) is observed, the w.t.d.f. $P(w_{FCFS} \leq t)$ - which is independent of priority assignment - has a bader or better position in the set of curves $P(w_{iNONPRE} \leq t)$, resulting.

The assignment of urgency numbers to customers with different mean service times β_i has the same effect as the assignment of static priorities: small urgencies correspond to small priority numbers (high priorities) while large urgencies correspond to large priority numbers. Assigning small urgency numbers to customers with small mean service times β_i and large v_i's to customers with large β_i's in the model of Fig. 1 results in a bad position of $P(w_{FCFS} \leq t)$ compared to $P(w_{\leq N} \leq t)$ or compared to the set of curves $P(w_i \leq t)$. Just this urgency number assignment minimizes the common mean w.t. $W_{\leq N}$ and optimizes the c.m.w.t.d.f $P(w_{\leq N} \leq t)$. An inverse urgency number assignment results in the least favorable set of curves $P(w_i \leq t)$ compared to $P(w_{FCFS} \leq t)$. Now it is quite clear that $\bar{v}'$ in Eq.(2.5) cannot be identical to $\bar{v}$ which is approximately the case for equal β_i's.

The optimal urgency number assignment just introduced is used in the example 2 cf. Fig. 3. It can be seen that $P(w_{FCFS} \leq t)$ is less favorable compared to the set of curves $P(w_i \leq t)$ because their common w.t.d.f. $P(w_{\leq N} \leq t)$ is favorable compared to $P(w_{FCFS} \leq t)$. The opposite is true for the example 4, cf. Fig. 4: There the smaller β_i is, the larger the urgency v_i. In Fig. 4 the w.t.d.f. $P(w_{FCFS} \leq t)$ is situated favorable in the set of curves $P(w_i \leq t)$.

Note that the load to the RU model is the same in both examples 2 and 4. Only the assignment of urgencies differ. Hence the w.t.d.f. $P(w_{FCFS} \leq t)$ is the same for both examples which is not the case for the curves $P(w_i \leq t)$ and $P(w_{\leq N} \leq t)$. Only the assignment of the urgency numbers is different, which precisely is the reason why $\bar{v}'$ differs from $\bar{v}$. Our observation is that $\varepsilon = \bar{v} - \bar{v}'$ for all the examples studied was always very small (a few percent of $\bar{v}$). Therefore we decide to accept Eq.(2.6) with $\bar{v}'$ substituted by $\bar{v}$ as an approximation which leads to

$$\lim_{f \to 1} [Q_i(f) - Q(f)] = v_i - \bar{v} \quad . \tag{2.9}$$

2.2. Approximate computation of the w.t.d.f. for the FCFS and RU-NONPRE disciplines

The w.t.d.f. in the FCFS discipline may be computed by inversion of their Laplace transform which can be a very difficult task. For the examples introduced in this paper it can be shown that the w.t.d.f.'s $P(w_{FCFS} \leq t)$ found by simulation espe-

cially in the tail, can be approximated very well by a degenerated exponential d.f. (DM-d.f.)

$$P(\leq t) = 1 - (1 - p)\, e^{-\mu t} \quad . \tag{2.10}$$

By setting the first and second moments of $P(w_{FCFS} \leq t)$ equal to the corresponding moments of Eq.(2.10), the parameters μ and p can be shown to be

$$p = \frac{c^2 - 1}{c^2 + 1} \qquad \text{and} \qquad \mu = \frac{2}{W_{FCFS}\,(1 + c^2)} \quad , \tag{2.11}$$

where

$$c^2 = W^{(2)}_{FCFS} / W^2_{FCFS} - 1 \tag{2.12}$$

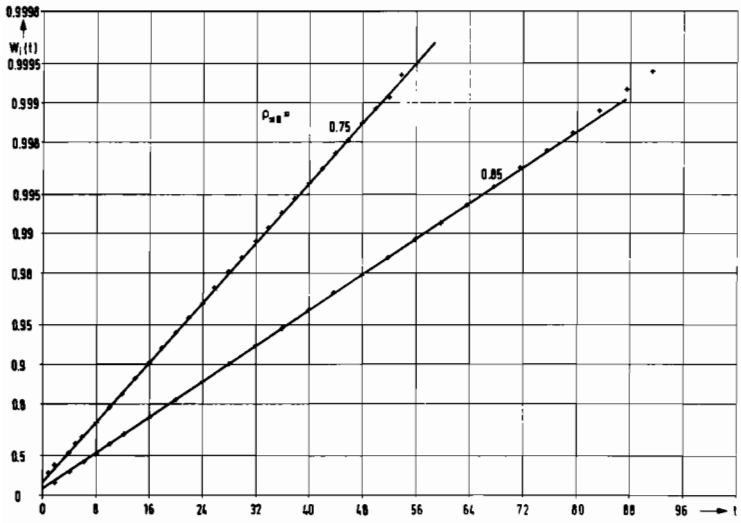
is the squared coefficient of variance and

$$W^{(2)}_{FCFS} = 2\, W^2_{FCFS} + \frac{\lambda_{\leq N}\, \beta^{(3)}_{\leq N}}{3(1 - \rho_{\leq N})} \quad . \tag{2.13}$$

W_{FCFS} and $W^{(2)}_{FCFS}$ are the first and second moments in the FCFS discipline, respectively. From this we have the approximation

$$P(w_{FCFS} \leq t) \approx (P \leq t) = 1 - (1 - p)\, e^{-\mu t} \quad . \tag{2.14}$$

We are now going to study the quality of Eq.(2.14) as an approximation for the w.t.d.f. $P(w_{FCFS} \leq t)$. Instead of computing the first two moments W_{FCFS} and $W^{(2)}_{FCFS}$ we will use the corresponding values found by simulation which somewhat compensates possible inaccuracies of our simulation results. In Fig. 7 through 9 the simulation results for the w.t.d.f. $P(w_{FCFS} \leq t)$ are shown using the parameters of the four examples in section 2. From this figures, in which the total offered traffic is a parameter, it can be seen that simulation (+) and approximation (straight line) fit together very well. This is especially true for the tails of the w.t.d.f.



Our goal is to approximate the tails of the w.t.d.f.'s in the RU discipline using Eq.(2.9). As a starting point we need a good approximation of the tail of the w.t.d.f. in the FCFS discipline. This prerequisite seems to be fulfilled.

Fig. 7: Waiting time d.f.'s of the model M/D/1/FCFS, cf. examples 2 and 4. Approximations by means of Eq.(2.10) (lines) and simulation results (+) are shown.
Parameter is the total offered traffic.

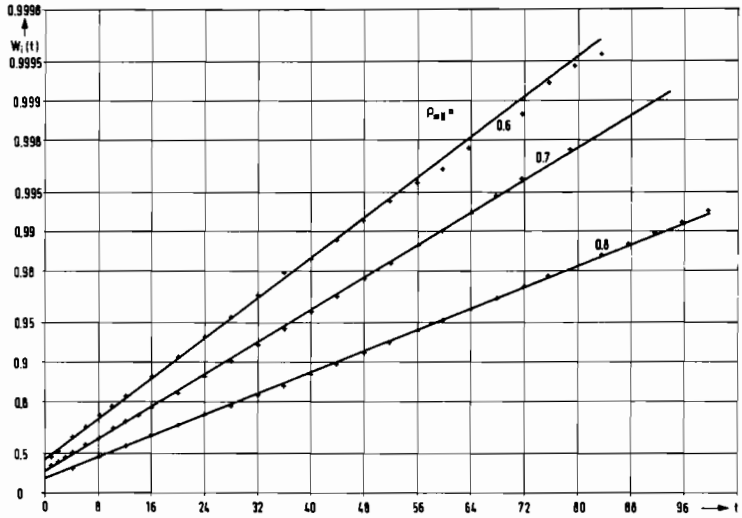


Fig. 8: Waiting time d.f.'s of the model M/M/1/FCFS, cf. example 1. Approximations by means of Eq.(2.10) (lines) and simulation results (+) are shown. The total offered traffic is a parameter.

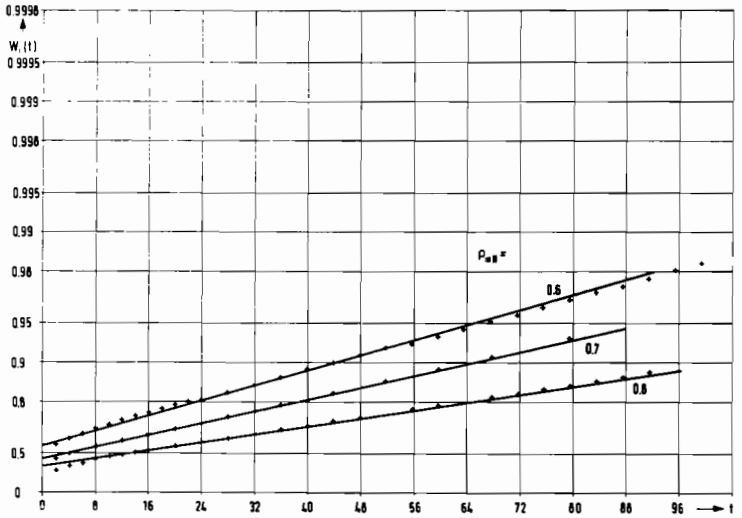


Fig. 9: Waiting time d.f.'s of the model M/G/1/FCFS, cf. example 3. Approximations by means of Eq.(2.10) (lines) and simulation results (+) are shown. The total offered traffic is a parameter.

2.2.1. Approximation of the tail of the w.t.d.f. in the RU-discipline

Using Eq.(2.9) it is possible to compute an expression $P'(w_i > t) = 1 - P'(w_i \leq t)$ which approximates the probability in the RU-NONPRE discipline of a waiting time w_i exceeding a large value of t

$$P'(w_i > t) \approx P(w_{FCFS} > t + \bar{\nu} - \nu_i) \quad . \tag{2.15}$$

From experience gained with simulation of the four examples one may call waiting times "large" if the relation $P(w_i > t) < 0.05$ is fulfilled. For medium and small sized waiting times it can be observed that, if

$$\begin{aligned} \bar{\nu} - \nu_i < 0 \qquad &\text{then} \qquad P(w_i > t) < P'(w_i > t) \\ \bar{\nu} - \nu_i > 0 \qquad &\text{then} \qquad P(w_i > t) > P'(w_i > t) \quad . \end{aligned} \tag{2.16}$$

Using Eqs.(2.9), (2.14) and (2.15) we get as an approximation

$$\lim_{t \to \infty} [P(w_i \leq t)] \approx 1 - (1 - p)\, e^{-\mu(\bar{\nu} - \nu_i)}\, e^{-\mu t} \quad . \tag{2.17}$$

Eq.(2.17) defines the tails of the w.t.d.f. in the RU-NONPRE discipline. Being adequate to parameter p in Eq.(2.14) we introduce a parameter p_i for type-i requests in the RU discipline and compute, by setting equal the right hand sides of both Eqs.(2.14) and (2.17),

$$p_i = 1 - (1 - p)\, e^{-\mu(\bar{\nu} - \nu_i)} \quad .$$

By insertion in Eqs.(2.17) we get

$$\lim_{t \to \infty} [P(w_i \leq t)] \approx 1 - (1 - p_i)\, e^{-\mu t} \quad .$$

Note that p and p_i are the probabilities which arise for $t = 0$.
From Simulation experiments we know that Eq.(2.17) is a good approximation, if either the total offered traffic is large, $\rho_{<N>} > 0.6$, or the urgency numbers have a small variance, or both. Otherwise, the w.t.d.f.'s in the RU discipline appear to be similar to those of the static priority discipline which is always the case for small waiting times t. The reason for this is that without the conditions mentioned, there is only a very small probability of missing deadlines for any type of request.
Figures 10 through 13 show both the simulation results (+,•) and approximate w.t.d.f.'s computed from Eq.(2.17) which appear as straight lines for the examples defined in section 2. It can be seen that for large waiting times, simulation and approximation agree very well. Deviations are typically below the 10 % range.

In Fig. 11 we can once again study the results of a different assignment of a set of urgency numbers ν_i to a set of mean service times β_i as defined by examples 2 and 4. The approximate w.t.d.f.'s are derived from the w.t.d.f. of the FCFS discipline and therefore are the same for both examples 2 and 3. As a consequence of the optimal urgency number assignment as defined in section 2.1.2 which is realized in the example 2, the real w.t.d.f.'s (•) compare favorably to the computed ones (lines). On the other hand the w.t.d.f.'s (x) of example 4, in which the ν_i are especially poorly assigned to the β_i, are less favorable than the computed ones. Similar results can be observed from Fig. 10 and 12 where the urgency numbers are assigned optimal. Note that the approximation by Eq.(2.1) does not take into account the urgency-number assignment. This should be the reason for the deviation between simulated and computed w.t.d.f.'s observed.

2.2.2. Approximation of the w.t.d.f. in the RU discipline for small waiting times

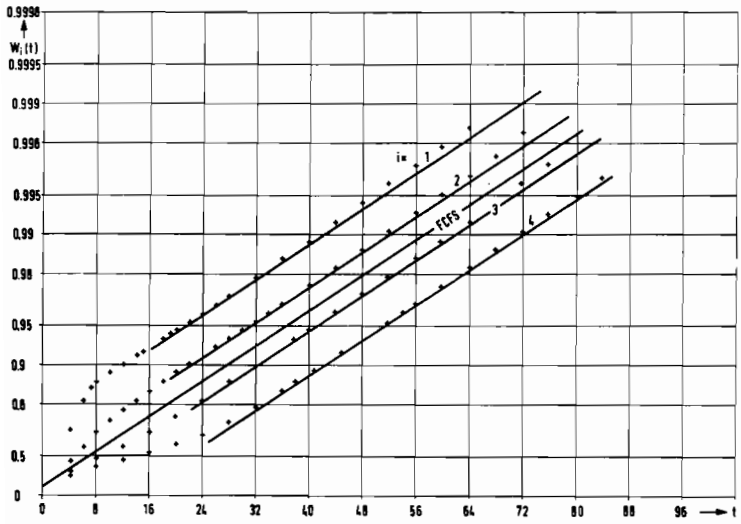


Fig. 10: Waiting time d.f.'s of the model M/D/1/RU-NONPRE, cf. example 2, approximated by means of Eq.(2.17) (lines) and simulation results (+). Additionally the FCFS approximation is shown. The total offered traffic is $\rho_{\leq N} = 0.85$.

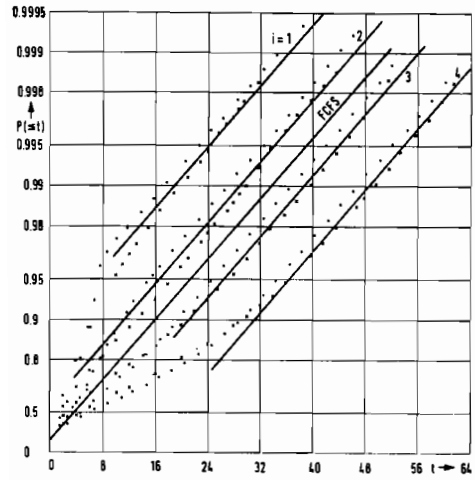


Fig. 11: Waiting time d.f.'s of the model M/D/1/RU-NONPRE, cf. examples 2 and 4, approximated by means of Eq. (2.17) (lines) and simulation results (+, •). Additionally the FCFS approximation is shown. The total offered traffic is $\rho_{\leq N} = 0.75$.

pline to compute the corresponding mean waiting time W_i of type-i requests

$$P(w_i > t) = \rho_{\leq N}\, e^{-\rho_{\leq N}\, t / W_i} \quad . \tag{2.18}$$

This simple approximation depends only on the total offered traffic $\rho_{\leq N}$ and W_i. Figures 14 through 16 show this approximation (lines) compared to simulation results (x) for examples 1, 2, and 3. The most unsatisfactory approximation appears to be for type-1 requests which was not needed in [1]. It can be seen that Eq.(2.18) in general cannot be called a good approximation. Even for small waiting times the approximated w.t.d.f. deviates from the simulated w.t.d.f. by up to 20 %.

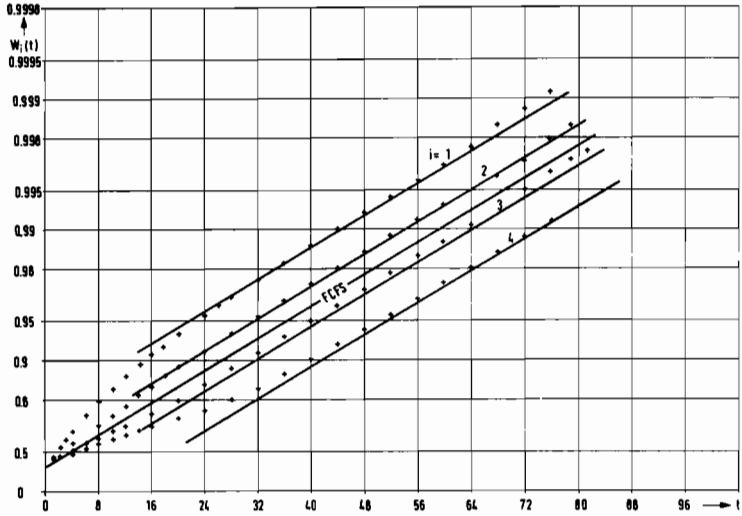


Fig. 12: Waiting time d.f.'s of the model M/M/1/RU-NONPRE, cf. example 1, approximated by means of Eq.(2.17) (lines) and simulation results (+). Additionally the FCFS approximation is shown. The total offered traffic is $\rho_{<N} = 0.75$.

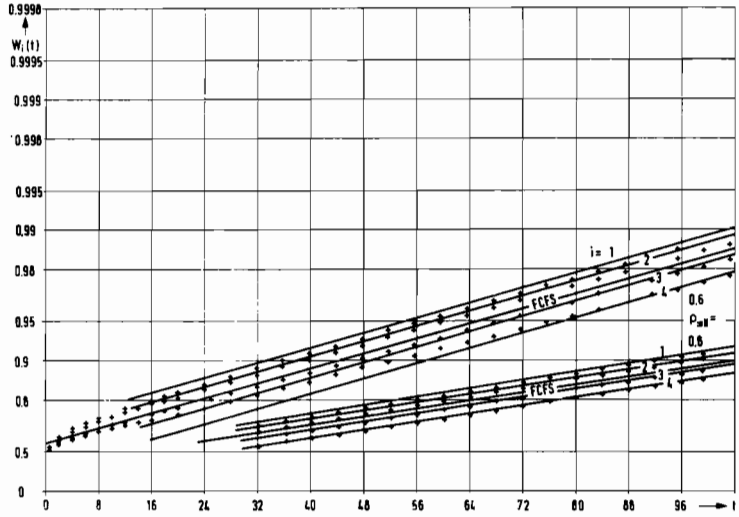


Fig. 13: Waiting time d.f.'s of the model M/G/1/RU-NONPRE, cf. example 3 of section 2, approximated by means of Eq.(2.17) and simulation results (+). Additionally the FCFS approximation is shown. The total offered traffic is a parameter.

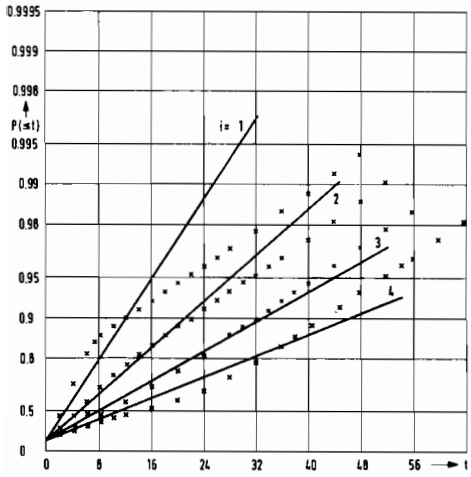


Fig. 14:
Waiting time d.f.'s of the model M/D/1/RU-NONPRE, cf. example 2, approximated by means of Eq.(2.18) (lines) and simulation results (x). The total offered traffic is $\rho_{\leq N} = 0.85$.

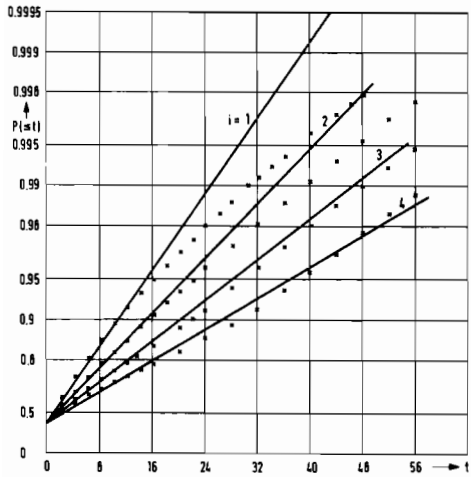


Fig. 15:
Waiting time d.f.'s of the model M/M/1/RU-NONPRE, cf. example 1, approximated by means of Eq.(2.18) (lines) and simulation results (x). The total offered traffic is $\rho_{\leq N} = 0.6$.

Nevertheless, the error produced by this approximation has a negligible effect on the computation of W_i, for simulation results agree well with the computed values of W_i, cf. [1].

2.3. Mean waiting time of requests which have missed their deadlines

From Eq.(2.17) the approximate probability of a type-i request missing its deadline (at $t = v_i$), presuming a sufficiently large t, can be computed

$$P(w_i > v_i) \approx (1 - p)\, e^{-\mu\bar{v}} \,. \tag{2.19}$$

The w.t.d.f. of requests missing their deadlines is approximately given by

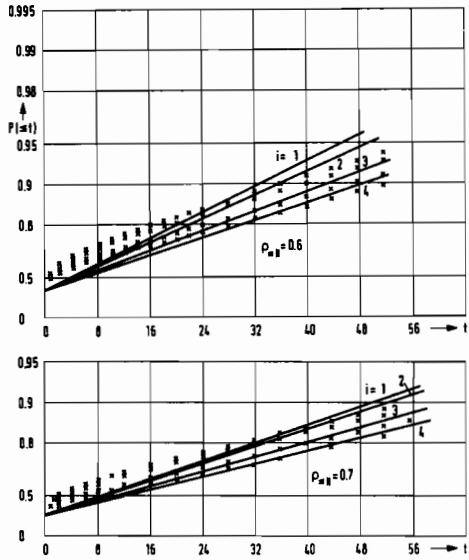


Fig. 16:
Waiting time d.f.'s of the model M/G/1/RU-NONPRE, cf. example 3, approximated by means of Eq.(2.18) (lines) and simulation results (x). The total offered traffic is a parameter.

$$P(w_i \le t \mid t > V_i) = \frac{P(w_i \le t) - P(w_i \le V_i)}{1 - P(w_i \le V_i)}$$

$$P(w_i \le t \mid t > V_i) = 1 - e^{\mu V_i}\, e^{-\mu t} \quad . \tag{2.20}$$

Apparently a DM-d.f. arises whose mean

$$E(w_i \mid t > V_i) = W_i^* = e^{\mu V_i} / \mu \tag{2.21}$$

only depends on μ and V_i. Note that μ can be computed from Eq.(2.11). The approximate mean waiting times W_i^* of requests having missed their deadlines differ approximately in the factor e^{V_i}.

3. COMPARISON OF W.T.D.F.'s OF THE RU-NONPRE AND STATIC PRIORITY DICIPLINES

The RU-NONPRE discipline may be thought of as an alternative to the static priority discipline. From [1] and [2] it is known that the greater the variance of the urgency numbers is the more the mean waiting time W_i in the RU discipline differs from the m.w.t. W_{FCFS} and the more the m.w.t.'s in the static priority discipline $W_{iNONPRE}$ are approximated. Now, depending on the set of V_i's, the question arises how the w.t.d.f.'s in the RU and static priority disciplines will differ. Moreover, it is interesting to know which of those disciplines better observes a given set of deadlines defined by the urgency numbers V_i.
In order to make such an investigation let us define a figure of comparison

$$\ddot{U} = \lambda^{-1}_{\le N} \sum_{i=1}^{N} \lambda_i P(w_i > V_i) \quad . \tag{3.1}$$

By computing the value of $\ddot{U}$ for different disciplines and a given set of V_i's and λ_i's, we are only able to compare the total probabilities of missing the deadlines given by the V_i's for all type-i requests together.

Recalling that the probabilities needed to compute Ü are available only from an approximation, Eq.(2.17), we decided to use simulation to clarify this question. For the four examples mentioned, we found regardless of the total offered traffic $\rho_{\leq N}$ that neither of the two disciplines is basically superior to the other. Depending on the case, either discipline can produce a lower probability Ü.

Generally speaking it seems that using static priorities and applying a priority assignment according to Eq.(2.8) the common probability $Ü = Ü_{stat}$ for missing the set of deadlines given by the v_i's is smaller than $Ü = Ü_{dyn}$ using the RU-discipline. The opposite is true if one deviates substantially from the optimal priority assignment, i.e. $Ü_{dyn} < Ü_{stat}$ arises.

Recalling from section 1 that

$$W_{istat}(v_i) = P(w_{istat} \leq v_i) = 1 - P(w_{istat} > v_i)$$

it is possible to transform the figure of comparison introduced in Eq.(1.5) to

$$V = \lambda_{\leq N}^{-1} \sum_{i=1}^{N} \lambda_i \left[P(w_{idyn} > v_i) - P(w_{istat} > v_i) \right]$$

$$V = Ü_{dyn} - Ü_{stat} \quad . \tag{3.2}$$

The relations defined by Eq.(1.6) still remain valid.
Simulation results in the static priority discipline of the examples 2 (broken lines) and 4 (solid lines) are shown in Fig. 17. For each example 1 million requests were processed in the simulation model. In example 2 the static priorities are op-

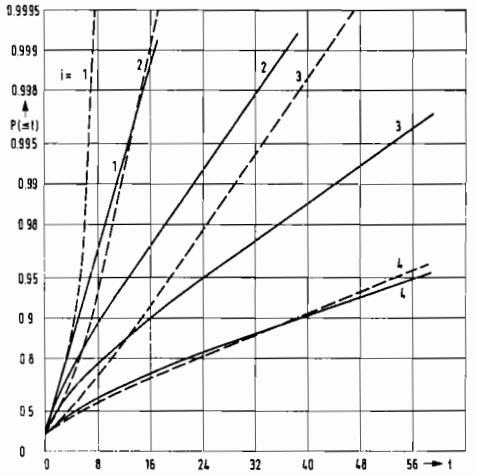


Fig 17:
Simulated waiting time d.f.'s of the model M/D/1 in the non-preemptive static priority discipline for the example 2 (dashed lines) and 4 (solid lines). The total offered traffic is $\rho_{\leq N} = 0.75$.

timally assigned while the opposite is true in example 4. The corresponding w.t.d.f.'s in the RU discipline are already known from Fig.11. For example 2 using the corresponding set of urgency numbers one computes $Ü_{stat} \approx 0.025 \approx Ü_{dyn} / 3$ which yields $V \approx 0.05$. For example 4 we find $Ü_{stat} \approx 0.5 \approx 3\, Ü_{dyn}$ which yields $V \approx -0.034$. Apparently, neither of the two disciplines can claim to always guarantee the minimum total probability Ü of missing the deadlines given by the set of the urgency numbers, independent of the parameter set applied.

This observation was also made with other values of $\rho_{\leq N}$ and other examples: example 1 with a total offered traffic $\rho_{\leq N} = 0.6$ yields $Ü_{dyn} = 0.172$, $Ü_{stat} = 0.147$, and

$V = 0.025$, i.e. the static priority discipline is superior.

For the example 3 we found for $\rho_{\leq N} = 0.6$ total probabilities $\ddot{U}_{dyn} = 0.215$, $\ddot{U}_{stat} = 0.172$; and from this $V = 0.043$ which indicates that here too the static priority discipline is superior.

In the examples considered, V has been observed to appear in the range of +5 % to -3.4 %. Recalling that V expresses mean values of probabilities of missing deadlines for all requests together we have substantial differences between the two disciplines compared. From our observations one can conclude that a priority assignment according to Eq.(2.8) results in $V > 0$, cf. Eq.(1.7).
One substantial difference between the static-priority and the RU-discipline is not taken into account by our comparison using V. This difference can be seen by comparing the individual probabilities of type-i requests in the same discipline: While the RU-discipline meets all deadlines of different requests given by the set of the ν_i's with nearly the same probability, this probability may differ by factors in the static priority discipline, presuming the same set of the ν_i's. In the RU-discipline all requests are handled on an equal basis while the opposite is true for the static priority discipline.

4. SOME TYPICAL PROPERTIES OF THE W.T.D.F. IN THE PREEMPTIVE RU-DISCIPLINE

The main differences between the preemptive and non-preemptive versions of the RU-discipline were already presented in the paragraph of section 1 preceding the abbreviations. Preemption allows a request being serviced ahead of time to be interrupted by a request whose deadline would otherwise be missed. Interruptions are allowed for this reason only.

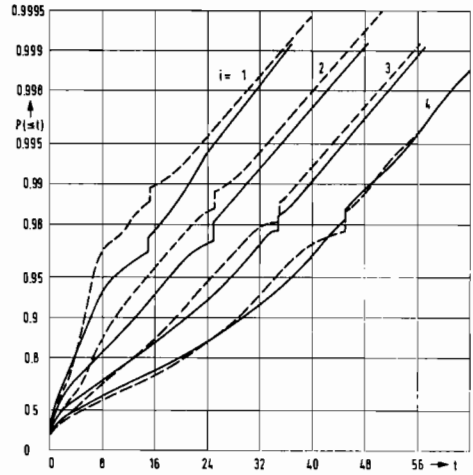


Fig. 18:
Simulated waiting time d.f.'s of the model M/D/1/RU-PRE for the examples 2 (solid lines) and 4 (dashed lines). The total offered traffic is $\rho_{\leq N} = 0.75$. The jumps appear at $t = \nu_i$.

The waiting-time d.f.'s in the RU-PRE discipline presuming the data of the examples 2 and 4 shown in Fig. 18 and example 3 in Fig. 19 were determined by simulation. Comparable results using the RU-NONPRE discipline are shown in Fig.'s 11 and 13. It can be seen that for the same total offered load, distinct differences in the waiting times of both disciplines arise. Especially worth noting is the jump at $t = \nu_i$ which results from interrupting prematurely started servicing for such requests which would otherwise miss their deadlines. The size of the jump exactly defines the probability for interrupting other types of requests. It can be observed that the w.t.d.f.'s in the tail in semilogarithmic scaling are parallel, as

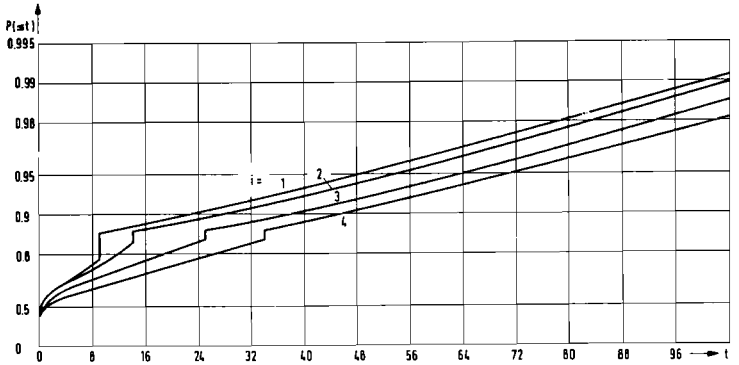


Fig. 19: Simulated waiting time d.f.'s of the model M/G/1/ RU-PRE, cf. example 3. The total offered traffic is $\rho_{\leq N} = 0.6$. The jumps appear at $t = v_i$.

was the case for the RU-NONPRE discipline (cf. [6]). Not for all examples considered the slope of these tails in the RU-PRE and RU-NONPRE disciplines was found to be the same (the same offered traffic $\rho_{\leq N}$ presumed).

It appears that interruptions are useful for increasing the probabilities of meeting deadlines if the coefficients of variance of the service time d.f.'s are large, i.e. $C_i > 1$ (cf. Fig.'s 13 and 19). For small C_i's our simulation supports the conjecture that interruptions of service time in favour of others having a larger expected service time is disadvantageous for meeting deadlines (cf. Fig.'s 11 and 18). The deadlines of some types of service requests are then met better with the RU-NONPRE discipline than with the RU-PRE discipline.

The common probability Ü for missing deadlines with either static priorities or the RU-PRE discipline was determined by simulation. With respect to Ü it can be seen that neither discipline is generally superior to the other, as was the case with non-preemptive priorities. Depending on the example presumed for such a comparison one of both disciplines appears to be superior.

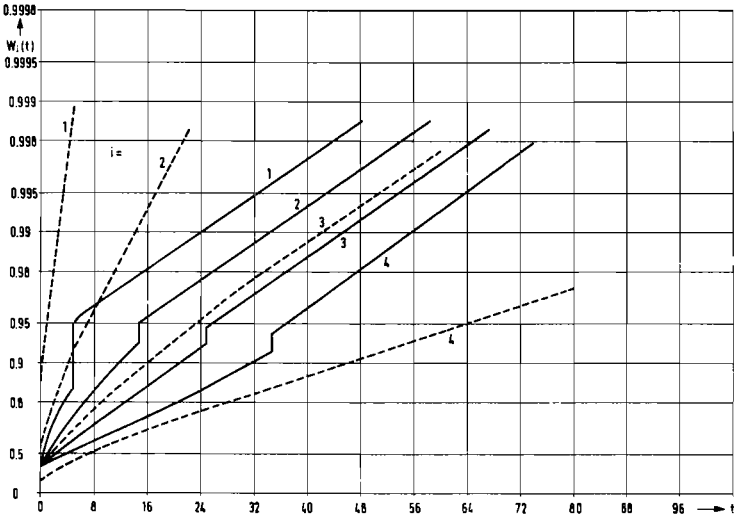


Fig. 20: Simulated waiting time d.f.'s of the model M/M/1 in both preemptive disciplines: static priorities (dashed lines) and RU (solid lines), cf. example 1. The total offered traffic is $\rho_{\leq N} = 0.6$. The jumps appear at $t = v_i$.

In the example shown in Fig.20 both sets of w.t.d.f.'s in the RU and static priority discipline can be compared. According to Eq.(1.7) we have V = 0.031. Upon comparing both versions of the RU-discipline, it was found that in the preemptive version the deadlines are always met somewhat better than in the non-preemptive version (cf. Fig.21). The probability of all types of request meeting their deadlines is, as was the case in the RU-NONPRE discipline, approximately the same.

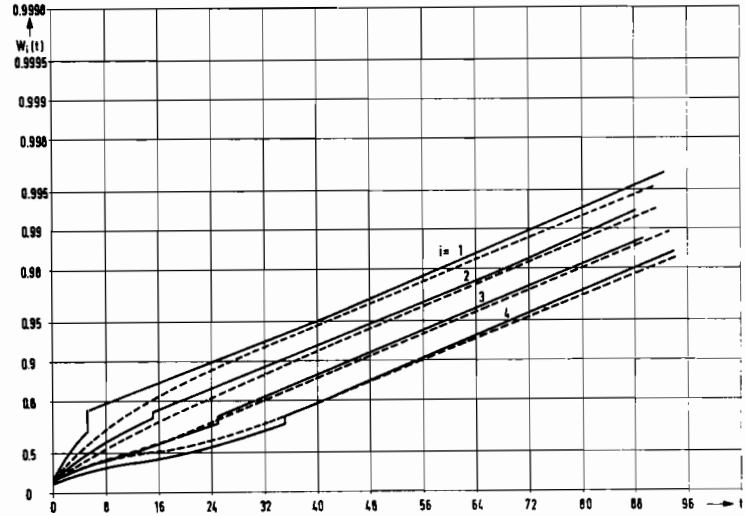


Fig. 21: Simulated waiting time d.f.'s of the model M/M/1 in the preemptive (solid lines) and non-preemptive (dashed lines) versions of the RU discipline, cf. example 1. The total offered traffic is $\rho_{\leq N} = 0.8$.

Literature:

[1] B. Walke: Improved bounds and an approximation for a dynamic priority queue. Third Int. Symp. Modell. and Performance Evaluat. Comp. Syst., Oct. 1977, Bonn, Germany. Proceed. published by North Holland Publishing Comp., Amsterdam, The Netherlands (1978) pp. 321 - 346.

[2] J.R. Jackson: Waiting time distributions for queues with dynamic priorities. Naval Res. Logistic. Quart., March (1962) Washington D.C., 9, pp. 31 - 46.

[3] R.W. Conway, W.L. Maxwell, L.W. Miller: Theory of scheduling. Addison Wesley Publishing Comp., Reading, Mass, (1967).

[4] B. Walke: Realzeitrechner-Modelle, Theorie und Anwendungen, R. Oldenbourg Verlag, Reihe Datenverarbeitung, München (1978).

[5] (not cited) E.B. Veklerov, I.M. Dukhovnyi: Queueing system with time-limited categorical priority. Probl. Inform. Transmiss., Vol.12, No. 1, Jan.-March (1976) pp. 73 - 79.

[6] H.M. Goldberg: Jackson's conjecture on earliest due date scheduling. Departm. Finance & Management Science, Faculty Busin. Administr. & Commerce, University of Alberta, Edmonton, Alberta, Canada T6G 2G1, Sept. (1978).

INFORMATION SYSTEMS

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

AN EXAMPLE FOR AN ADAPTIVE CONTROL METHOD PROVIDING DATA BASE INTEGRITY

A. Benczúr and A. Krámli
Computer and Automation Institute of
Hungarian Academy of Sciences
Budapest, Hungary

An adaptive version of system recovery is investigated under the condition that the failure process is an inhomogeneous Poisson process of simple structure. The admissible class of strategies is characterized by 3 parameters. An explicite formula is given for the average cost which depends on the parameters.

INTRODUCTION

The reliability of complicated software systems providing interactive real time access has more and more importance when the size of them increases. Among the numerous aspects of reliability (see e.g. [5]) the large data base systems being in the centre of applications cause the most specific problems. The methods of providing the integrity of data base systems (recovery procedures) has recently been developed (see e.g. [8]) therefore the dynamical investigations, the probabilistic analysis of the cost of these procedures has also become necessary. The authors' first step in this direction was done on the basis of the experiences gathered from the realization of a concrete data base management system [3].

In this paper there is given a model of the data base management system in terms of notions similar to those in the theory of automata. This model is appropriate for describing the problems of integrity because the range of effect of the random failures is restricted. This work shows the possibility of the application of central limit theorem for determining the probability distribution of the extra run time caused by failures. A. Benczúr's Ph. D. thesis [2] extends the limit theorem method. In addition, it contains comparisons of the stochastic investigations in the literature; first of all the results of K.M. Chandy and E.Gelenbe

[6], [7], [9], [10].

Here we do not repeat the descriptive models treated in [2] since the great part of the dynamical investigations is concerned with the method of generation of control points according to Davies' paper [8] this being the basic method of system recovery .

The main goal of the present talk is to give an explicite formula for the average cost of a simple adaptive recovery procedure. The stochastic model applied here was suggested by a result of E. Gelenbe [10] , and a problem stating lecture of M. Arató [1].

THE BASIC MODEL

The system has the following 4 possible states:

I., normal processing;
II., generating a control point (dump);
III. regenerating a control point after a failure (load);
IV., repetition of the tasks processed since the nearest control point to a failure (recovery).

Let us denote by μ_t (ν_t) the failureless run time necessary for a dump (load) which begins at the moment t. (μ_t and ν_t are determined by the past of the system . Further on, let $g(t,t_1)$ ($h(t,t_1)$) denote the run time consumed in the failureless time interval (t,t') while the system is staying in state I (in state IV) for the normal processing (recovery) on this time interval. (Notice that in these cases a time shared computer system deals with other task too, moreover during the normal processing it can wait for a new task or transaction.)

If a failure occurs when the system is in one of the states I and IV the corresponding control point must be loaded (state III) and the processed tasks must be repeated. Let us assume that a failure occured at t_0+t , the last control point was generated at t_0, and the system was in state I between t_0 and t_0+t in the disjoint time intervals

$$(t_0, t_0+t_1'), \dots (t_{n-1}, t_{n-1}+t_n') \quad \text{where} \quad (t+t_0 = t_{n-1}+t_n')$$

So the run time required for normal processing on the interval (t_0, t_0+t) is equal to $\sum_{i=1}^{n} g(t_{i-1}, t_{i-1}+t_i')$.
If the system begins the recovery at $t'' > t_0+t$ then the run time required satisfies the condition

$$h(t'', t''+T) = H\left(\sum_{i=1}^{n} g(t_{i-1}, t_{i-1}+t_i')\right)$$

assuming that failures do not occur on $(t'', t''+T)$, where H is a given function. Naturally during the recovery failures may occur and control points can be generated. The fundamental problem is to minimize the time spent in states II, III and IV (or the total cost of the corresponding procedures) relative to the time spent in state I. This may be done by choosing appropriately the moments $T_1, T_2, \ldots$ of dumps depending on the stochastic behaviour of the failure process, and on the intensity function $g(t, t_1)$
K.M. Chandy and his coauthors [6], [7] investigate deterministic dump strategies. They assume that the states II, III, and IV are points in time, and they often calculate with costs proportional to their expected times.
They analyse the following cases:

a) - The failure process is a homogeneous Poisson process,
$H(x) = bx$, $g(t, t_1) = a(t_1 - t)$
- the cost of dumps and loads is constant,
- failures may occur only during the normal processing.
The optimal equidistant dump strategy is determined.

b) - As a) but failures may occur during the dumps and loads too.
A numerically solvable equation is given for the optimal equidistant dump strategy.

c) - The failure process is an inhomogeneous Poisson process with a periodic intensity function $\lambda(t)$,
- $g(t, t_1) = \int_t^{t_1} \lambda(x)\, dx$,
- the functions $g(\cdot)$ and $\lambda(\cdot)$ has the same period d.
A recursive system of equation is given for the optimal arrangement of the control points on the interval d.

In the investigations proceeded by E. Gelenbe [9], [10] the time is dominant. The failure process is a homogeneous Poisson process. The run times μ_t required by the dumps form a sequence $\{\mu_i\}$ of independent identically distributed random variables, $g(t, t_1) = t_1 - t$ $h(t, t_1) = t_1 - t$ and $H(x)$ is an increasing function. The distances between the control points measured on the time scale of the normal processing form a sequence $\{\Delta_i\}$ of independent identically distributed random variables. The failure process, the processes $\{\mu_i\}$ and $\{\Delta_i\}$ are independent of each other. The

stationary probabilities of being in states I-IV. are determined. The stationary probability of being in the state I is maximal if and only if the distribution function of Δ_i is concentrated to a constant. If the system is considered as a queueing system then there is given the necessary and sufficient condition of ergodicity. The earlier results of the authors of this talk refer to the case, when the failure process and $\{\Delta_i\}$ are general renewal processes under the following conditions

$$\mu_t = 0, \quad \nu_t = 0$$
$$g(t,t_1) = t_1 - t$$
$$h(t,t_1) = t_1 - t$$
$$H(x) = x .$$

In papers [3] and [2] the limit distribution of the extra run time caused by the failures is determined when the time of normal processing tends to the infinity. These results enable us to determine the average cost of a recovery procedure even if the cost is a nonlinear function of the extra run time.

The validity of the central limit theorem is proved in the following cases

- the distances between the control points are constant,
- the time is discrete, the distances between the control points are bounded, there exist some higher moments of the distribution of the intervals between the failures,
- the time is continuous and the intervals between the failures have Γ-distribution.

In the present talk we use Chandy's approximation, assuming that μ_t and ν_t are points and we calculate only with their costs. This assumption is reasonable from practical point of view, because during the relatively fast dumps and loads we use much more facilities of the computer system than during the normal processing. The optimal control point generation strategy is connected with the classical disorder problem. M. Arató stated the question [1], whether the results on the disorder problem would have applications in computer sciences. Naturally we do not solve the general problem; we are seeking the optimal strategy in a class of admissible strategies described by 3 parameters. An explicite formula is given for the relative cost of the strategy as a function of these parameters.

THE AVERAGE COST OF AN ADAPTIVE CONTROL POINT GENERATION STRATEGY

Let $\{\tau_n\}$ denote the failure process: τ_1 is equal to the time interval until the first failure, while τ_n is equal to the time interval between the $n-1$st and n-th failure. Let $\{\nu^{(i)}\}$ be a sequence of independent geometrically distributed random variables (i.e. $P(\nu^{(i)} = k) = pq^{k-1}$; $k = 1,2,\ldots$; $q = 1-p$). In the course of this talk we suppose that $\{\tau_n\}$ satisfies the following conditions:

(i) $\{\tau_1,\ldots,\tau_{\nu(1)},\ \tau_{\nu(1)+2},\ldots,\tau_{\nu(1)+\ldots+\nu(k)+k+1},\ldots\}$ forms a sequence of independent exponentially distributed random variables with parameter λ_1.

(ii) $\{\tau_{\nu(1)+1},\ldots,\ \tau_{\nu(1)+\ldots+\nu(k)+k},\ldots\}$ forms a sequence of independent exponentially distributed random variables with parameter $\lambda_2 < \lambda_1$

(ii) the sequences $\{\nu^{(i)}\}$, $\{\tau_1,\ldots,\tau_{\nu(1)},\tau_{\nu(1)+2},\ldots\}$ and $\{\tau_{\nu(1)+1},\ldots,\tau_{\nu(1)+\ldots+\nu(k)+k}\}$ are independent of each other. (See Figure 1.)

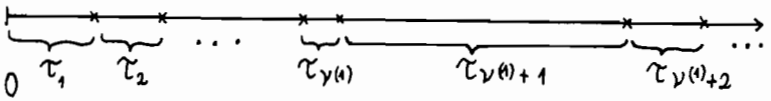


Figure 1.

Remark: The sequence of time intervals $\theta_1 = \sum_{j=1}^{\nu(1)} \tau_j,\ldots,$

$$\theta_k = \sum_{j=k+\nu(1)+\ldots+\nu(k-1)}^{k+\nu(1)+\ldots+\nu(k)} \tau_j$$

forms a sequence of independent exponentially distributed random variables with parameter $p\lambda_1$.

The integrity of the data base is provided according to our model analysed in paper [3]:

$$g(t,t_1) = h(t,t_1) = t - t_1$$

$$H(x) = x$$

$$\mu_t = K_1 \ , \ \nu_t = K_2$$

Failures may be occured during the repetition of tasks too. (See Figure 2.)

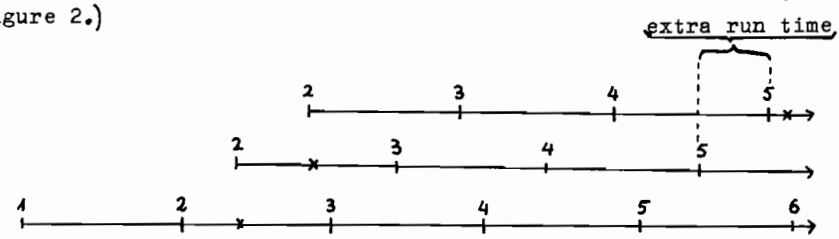


Figure 2.

We say that the system is in state h (heavy) at the moment t iff t belongs to one of the intervals θ_k (otherwise it is in state $\bar{h}$). Because of the conditions (i)-(iii) the time T passed since the last failure contains the only information about the state of the system. Therefore it is reasonable to consider only the following class of admissible decision strategies described by 3 parameters: if $T > d$ then the system generates control points by equidistant steps t_2 and after a failure it generates control points by steps $t_1 < t_2$. More precisely, we say that the decision $d(t)$ takes two values H and $\bar{H}$: after every failure $d(t) = H$ and after every failureless interval of length d $d(t) = \bar{H}$. The set $\{ t : d(t) = H \}$ consists of intervals beginning with failure. The recovery procedure after such a failure repeats the tasks from the last control point, but generates control points by step t_1 independently of the former density of the control points. Let us suppose that $d(t)$ changes it's value to $\bar{H}$ at a moment t. The last control point was made at the moment $t - d + [\frac{d}{t}]t_1$ so the system generates the next control point at $t - d + [d/t_1]t_1 + t_2$ if the interval $(t, t-d+[d/t_1]t_1 + t_2]$ is failureless. (See Figure 3.)

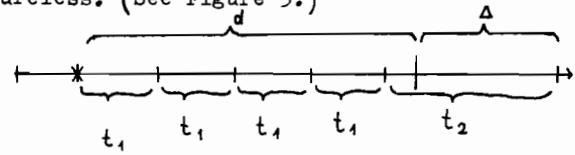


Figure 3.

Generally the optimal strategy does not belong to the class described above. However it has special interest because an explicite formula can be derived for the average cost $C(t_1, t_2, d)$ For this purpose let us introduce the following stochastic process:

$$\eta(t) = \begin{cases} 1 \text{ if the system is in the state } h \text{ and } d(t) = H \\ 2 \quad " \quad h \text{ and } d(t) = \bar{H} \\ 3 \quad " \quad \bar{h} \text{ and } d(t) = H \\ 4 \quad " \quad \bar{h} \text{ and } d(t) = \bar{H} \end{cases}$$

Lemma 1. The process $\eta(t)$ is semi-markovian, and the embedded Markov chain η_n has the following transition probabilities:

$$(1) \qquad \begin{aligned} p_{1,2} &= \frac{e^{-\lambda_1 d}}{1 - q(1 - e^{-\lambda_1 d})}, \\ p_{1,3} &= 1 - p_{1,2}, \\ p_{2,1} &= 1 - p, \quad p_{2,3} = p \\ p_{3,1} &= 1 - e^{-\lambda_2 d}, \quad p_{3,4} = e^{-\lambda_2 d} \\ p_{4,1} &= 1 \end{aligned}$$

and $p_{i,j} = 0$ for every other transition $i \to j$.

The semi-markov property is a straightforward consequence of conditions (i) - (iii) and the definition of $\eta(t)$. Among the transition probabilities only $p_{1,2}$ and $p_{1,3}$ are to be explained:

$$(2) \qquad \begin{aligned} p_{1,2} &= P(\mu \le \nu) \\ p_{1,3} &= P(\mu > \nu) \end{aligned}$$

where μ and ν are independent geometrically distributed random variables with parameters $e^{-\lambda_1 d}$ and p respectively.

By technical reasons we consider the trajectories of $\eta(t)$ to be continuous from left.

Let δ_n, μ_n, τ_n and C_n denote the sojourn time, the number of failures, the extra run time caused by failures and the number of dumps and loads respectively on the (open from left and closed from right) time interval $\{t : \eta(t) = \eta_n\}$

Lemma 2.
The discrete time parameter stochastic processes $\{\delta_n\}$ $\{\mu_n\}$ $\{\tau_n\}$ and $\{C_n\}$ form a

sequence of independent random vectors under the condition that the trajectory of $\{\eta_n\}$ is fixed. Naturally its components depend on each other. Moreover their conditional distribution depends only on η_{n-1} and η_n.

Proof. The statement of the lemma is an obvious consequence of conditions (i) - (iii), the lemma 1 and the definition of processes $\{\delta_n\}$ $\{\mu_n\}$ $\{\tau_n\}$ and $\{c_n\}$

Set

$$E^{(\delta)}_{i,j} = E(\delta_n \mid \eta_{n-1} = i\ ,\ \eta_n = j)$$
$$E^{(\mu)}_{i,j} = E(\mu_n \mid \eta_{n-1} = i\ ,\ \eta_n = j)$$
$$E^{(\tau)}_{i,j} = E(\tau_n \mid \eta_{n-1} = i\ ,\ \eta_n = j)$$
$$E^{(c)}_{i,j} = E(c_n \mid \eta_{n-1} = i\ ,\ \eta_n = j)$$

and let P_1, P_2, P_3, P_4 be the stationary probabilities of the ergodic aperiodic Markov chain η_n.

Theorem 1. The average cost $C(t_1, t_2, d)$ of the recovery procedure can be calculated by the following formula:

$$C(t_1,t_2,d) = \lim_{N\to\infty} \frac{\sum_{n=1}^{N}((K_1-K_2)\mu_n + \tau_n + K_1 c_n)}{\sum_{n=1}^{N}(\delta_n - \tau_n)} =$$

$$= \frac{\sum_{i,j=1}^{4} P_i P_{ij} [(K_1-K_2) E^{(\mu)}_{i,j} + E^{(\tau)}_{i,j} + K_1 E^{(c)}_{i,j}]}{\sum_{i,j=1}^{4} P_i P_{ij} (E^{(\delta)}_{i,j} - E^{(\tau)}_{i,j})} \tag{3}$$

Proof. Theorem 1 is a straightforward consequence of the strong law of large numbers and the ergodic theorem for Markov chains. In the sequel we give explicite formulas for the conditional expectations $E^{(\delta)}_{i,j}$, $E^{(\mu)}_{i,j}$, $E^{(\tau)}_{i,j}$ and $E^{(c)}_{i,j}$.

First we introduce some general notations.

Set

$$h_t(x) = x - nt \quad \text{if} \quad nt \le x < (n+1)t\ ,\ n = 0, 1, \ldots$$

Let $F_{i,j}(x)$ be the conditional distribution function of the time intervals between the failures occured in the time interval $\{t : \eta(t) = \eta_k\}$ under the condition that $\eta_{k-1} = i$ and $\eta_k = j$.

If the system begins its work after a the n-th failure from a

control point, and it generates control points by step t then the expected extra run time $E_t^{F_{i,j}}$ caused by the $n+1st$ failure has the form

$$E_t^{F_{i,j}} = \int_0^\infty h_t(x)\, dF_{i,j}(x) .$$

In our case $F_{i,j}(x)$ has two types:

$$\text{(I)} \qquad F^{(I)}_{(x)} = \begin{cases} \dfrac{1-e^{-\lambda x}}{1-e^{-\lambda d}} & \text{for } x \le d \\ 1 & \text{for } x > d \end{cases}$$

$$\text{(II)} \qquad F^{(II)}_{(x)} = 1 - e^{-\lambda x}$$

Set

$$E^{(\lambda)}_{t,d} = \int_0^d h_t(x)\, \frac{\lambda e^{-\lambda x}}{1-e^{-\lambda d}}\, dx \quad .$$

If $n = \left[\frac{d}{t}\right]$ then

$$E^{(\lambda)}_{t,d} = (1-e^{-\lambda d})^{-1}\left(\frac{1}{\lambda} - \frac{t e^{-\lambda t}(1-e^{-\lambda n t})}{1-e^{-\lambda t}} - \left(\frac{1}{\lambda} + d - nt\right)e^{-\lambda d}\right)$$

Observe that

$$E^{(\lambda)}_{t,t} = E^{(\lambda)}_{t,\infty} = \frac{1}{\lambda} - \frac{t\, e^{-\lambda t}}{1-e^{-\lambda t}} .$$

In a similar way we determine the expected number $e_t^{F_{i,j}}$ of dumps, and loads between the n-th and $n+1st$ failure. Notice that the $n+1st$ failure causes a load :

$$e_t^{F_{i,j}} = \sum_{k=0}^{\infty} \int_{kt}^{\infty} dF_{i,j}(x) .$$

In our special cases (I) and (II) :

$$e^{(\lambda)}_{t,d} = (1-e^{-\lambda d})^{-1}\left(\frac{1-e^{-\lambda(n+1)t}}{1-e^{-\lambda t}} - (n+1)\, e^{-\lambda d}\right)$$

$$e^{(\lambda)}_{t,\infty} = \frac{1}{1-e^{-\lambda t}}$$

Now we give the explicite formulas for every possible transition:

$$E^{(\delta)}_{1,2} = E^{(\mu)}_{1,2} \cdot E^{(\lambda_1)}_{d,d} + d$$

$$E^{(\mu)}_{1,2} = \left(q\left(1-e^{-\lambda_1 d}\right)\right)\left(1-q\left(1-e^{-\lambda_1 d}\right)\right)^{-1}$$

$$E_{1,2}^{(t)} = E_{1,2}^{(\mu)} E_{t_1,d}^{(\lambda_1)}$$

$$E_{1,2}^{(c)} = E_{1,2}^{(\mu)} e_{t_1,d}^{(\lambda_1)} + \lceil d/t_1 \rceil$$

(Notice that the number of failures μ during the transition from the state 1 to the state 2 is geometrically distributed:

$$P(\mu = k) = (1 - q(1 - e^{-\lambda_1 d}))\, q^k (1 - e^{-\lambda_1 d})^k, \quad k = 0, 1, \ldots)$$

Similarly we get

$$E_{1,3}^{(\delta)} = E_{1,3}^{(\mu)} E_{d,d}^{(\lambda_1)}$$

$$E_{1,3}^{(\mu)} = (1 - q(1 - e^{-\lambda_1 d}))^{-1}$$

$$E_{1,3}^{(t)} = E_{1,3}^{(\mu)} E_{t_1,d}^{(\lambda_1)}$$

$$E_{1,3}^{(c)} = E_{1,3}^{(\mu)} e_{t_1,d}^{(\lambda_1)}$$

(Here $P(\mu = k) = (1 - q(1 - e^{-\lambda_1 d}))\, q^{k-1} (1 - e^{-\lambda_1 d})^{k-1}$, $k = 1, \ldots$)

$$E_{2,1}^{(\delta)} = E_{2,3}^{(\delta)} = \frac{1}{\lambda_1}$$

$$E_{2,1}^{(\mu)} = E_{2,3}^{(\mu)} = 1$$

$$E_{2,1}^{(t)} = E_{2,3}^{(t)} = e^{-\lambda_1 \Delta} E_{t_2,\infty}^{(\lambda_1)} + (1 - e^{-\lambda_1 \Delta})(E_{\Delta,\Delta}^{(\lambda_1)} + d - nt_1),$$

where $\Delta = nt_1 + t_2 - d$. (See Figure 3.)

$$E_{2,1}^{(c)} = E_{2,3}^{(c)} = 1 + e^{-\lambda_1 \Delta} e_{t_2,\infty}^{(\lambda_1)}$$

$$E_{3,1}^{(\delta)} = E_{d,d}^{(\lambda_2)}$$

$$E_{3,1}^{(\mu)} = 1$$

$$E_{3,1}^{(\tau)} = E_{t_1,d}^{(\lambda_2)}$$

$$E_{3,1}^{(c)} = e_{t_1,d}^{(\lambda_2)}$$

$$E_{3,4}^{(\delta)} = d$$

$$E_{3,4}^{(\tau)} = E_{3,4}^{(\mu)} = 0$$

$$E_{3,4}^{(c)} = \left[\frac{d}{t} \right]$$

$$E_{4,1}^{(\delta)} = \frac{1}{\lambda_2}$$

$$E_{4,1}^{(\mu)} = 1$$

$$E_{4,1}^{(\tau)} = e^{-\lambda \Delta} \left(\frac{1}{\lambda_2} - \frac{t_2 e^{-\lambda_2 t_2}}{1 - e^{-\lambda_2 t_2}} \right) + (1 - e^{-\lambda_2 \Delta}) \left(E_{\Delta,\Delta}^{(\lambda_2)} - d - n t_1 \right)$$

$$E_{4,1}^{(c)} = 1 + e^{-\lambda_2 \Delta} e_{t_2,\infty}^{(\lambda_2)}$$

The following tables illustrate the behaviour of the function $C(t_1, t_2, d)$ if $\lambda_1 = 1$, $\lambda_2 = 0.1$, $\rho = 0.1$ and $K_1 = K_2 = 1$

d=3

t_2 \ t_1	0.9	1.0	1.1	1.2
2.0	1.874	1.858	1.853	1.857
2.5	1.866	1.847	1.849	1.850
3.0	1.872	1.852	1.860	1.860
3.5	1.887	1.864	1.880	1.877

d=4

t_2 \ t_1	0.9	1.0	1.1	1.2
2.0	1.885	1.868	1.853	1.857
2.5	1.873	1.855	1.841	1.844
3.0	1.876	1.856	1.844	1.846
3.5	1.885	1.865	1.855	1.857

d=5

t_2 \ t_1	0.9	1.0	1.1	1.2
2.0	1.899	1.881	1.862	1.866
2.5	1.887	1.868	1.849	1.852
3.0	1.888	1.868	1.850	1.852
3.5	1.896	1.875	1.858	1.860

Let $C(t_1, t_2)$ denote the average cost of the recovery procedure when the state of the system is known. If $K_1 = K_2$ then

$$C(t_1,t_2) = \frac{p\left(E^{(\lambda_1)}_{t_1,t_1} + e^{(\lambda_1)}_{t_1,\infty}\right) + E^{(\lambda_2)}_{t_2,t_2} + e^{(\lambda_2)}_{t_2,\infty}}{p\left(\frac{1}{\lambda_1} - E^{(\lambda_1)}_{t_1,t_1}\right) + \frac{1}{\lambda_2} - E^{(\lambda_2)}_{t_2,t_2}}$$

As a comparison the following table illustrates the behaviour of $C(t_1,t_2)$:

t_2 \ t_1	0.9	1	1.1	1.2
2.0	1.457	1.423	1.398	1.381
2.5	1.434	1.399	1.374	1.357
3.0	1.428	1.392	1.367	1.350
3.5	1.431	1.396	1.371	1.393

Notice that the average cost in the uncontrolled case has the minimum:

$$\min_t C(t,t,d) \approx 1.947 .$$

References

[1] M. Arató: Statistical problems connected with queuing models for operating systems, Proc. Colloquium on Point Processes and Queuing Theory, Keszthely (1978 to appear).

[2] A. Benczur: Integrity of data base management systems in Hungarian Ph. D. thesis (Budapest 1977).

[3] A. Benczur, A. Krámli: On data base integrity, Acta Cybernetica 4.1 Szeged (1978) 181-185.

[4] A. Benczur, A. Krámli, J. Pergel : A central limit theorem for the extra run time caused by failures in a data base

management system, Proc. Colloquium on Point Processes and Queuing Theory, Keszthely (1978 to appear)

[5] G. A. Champine: What makes a system reliable? Datamation 24.9 (1978) 194-210.

[6] K. M. Chandy et al. : Analytic models for roll-back and recovery recovery strategies in data base systems, IEEE Trans. on Software Engineering 1.1 (1975) 100-110.

[7] K. M. Chandy: A survey of analytic models of roll-back and recovery strategies, Computer 8.5 (1975) 40-47.

[8] C. T. Davies Jr.: Data processing spheres of control, IBM Systems J. 17.2 (1978) 179-198.

[9] E. Gelenbe , O. Derochette: On the stochastic behaviour of a computer system under intermittent failures, Modelling and Performance Evaluation of Computer Systems (North-Holland, Amsterdam, 1976) .

[10] E. Gelenbe: Existence and uniqueness of stationary distributions in a model of roll-back recovery, Proc. International Symposium onNew Trends in System Analysis IRIA (1976).

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

A PREDICTIVE PERFORMANCE EVALUATION TECHNIQUE FOR INFORMATION SYSTEMS

Frank W. Allen
Management Information Systems Department
University of Arizona
Tucson, Arizona

The complexity and uniqueness of information systems restrict the number of qualitative performance prediction techniques available. This work isolates the file organization as a submodule of the information storage and retrieval system. A model for a general file organization is described by a flow graph. Algorithms are derived and applied to the flow graph to define an analytical model. The linear equation derived results in comparative response time values through a variance of parameters for prediction analyses. The analysis of the model is also shown to be applicable to larger systems.

INTRODUCTION

A model represents an organization of information about a system. Modeling of the system includes (1) defining the entities, attributes, and activities of the system, and (2) defining the relationships involved in the activities. The purpose of a system study dictates the nature of the model. Investigation of the functional relationships within the system provides a range of alternatives from which to choose attribute values. These relationships are derived approximations from queing theory, empirical studies, or estimation (Svobodava (1976)). Prediction of future system response is deduced based on analyses of the model's behavior.

Models used in system studies can be classified into physical models (analog) or mathematical models. Simulation is a common technique for modeling physical systems. Analytical methods are often used to construct mathematical models. Simulation provides excellent results of specific event-driven systems, but the cost of constructing the simulation may be restrictive. Validation of the simulation may also prove problematic. An analytical model is especially useful for scaling performance to provide comparative performance statistics. The varying of parameters is easily performed to establish first order approximations.

Following the classification guidelines set by Lucas (1971), performance evaluation is conducted for three general purposes: selection evaluation, performance evaluation, and performance monitoring. Selection evaluation utilizes an existing prototype of a proposed system to decide between system configurations. Benchmarking is the classic example of selection evaluation. However, the complexity and uniqueness of information systems has reduced the importance of benchmarking. Performance monitoring examines an operational system for purposes of optimization or deciding whether to upgrade. The goal of performance projection is the estimation of the performance of a system which does not yet exist. Simulation or analytical models are performance projection techniques.

Drummond (1969) types performance evaluation into classification, time evaluation, or comparison. System expectations are qualitatively defined by comparison. Presented are algorithms for measuring the discrete capabilities of a system with respect to time. The derived values provide measures of goodness through relative performance. The performance of the model is described by an analytical model in which the dependent variable is the expected response time. A Management Information System (MIS) is composed of a number of independent functional modules which collectively describe the performance. This paper isolates the file organization of an information storage and retrieval system as the MIS module to be investigated. Analytical models of the physical database structure have previously been forwarded by Cardenas (1973), Yao (1977) and Ghosh and Tuel (1976) to estimate the performance of proposed database management systems.

A functional module is defined by a hierarchy of levels identifying discrete tasks. A file organization is a two-level hierarchy functionally divided by algorithms to create and search a primary memory directory and a secondary storage data structure. Parameters are defined for the discrete capabilities of a general file organization in terms of probability distributions and discrete time intervals. The flow of information is presented as a flow diagram. This flow graph is the basis of the model for further investigation. The probabilities of transition between nodes of the graph and determination of time intervals for alternate logical paths of data flow lead to a Markov chain analysis.

The reduction of the information flow graph produces a linear equation in the regular expression algebra by means of simple algorithms. A Z-transform on the time intervals identifies time delays in the equation and results in a first order differential equation. The first moment of this equation provides the expected time delay of the model. Varying the parameters of this equation within realistic ranges results in comparative statistics for performance projections. Interpretation of the results benefits the management decision-making capability. This is supported by viewing the general analytical model.

Analysis of the model is also shown to be applicable to larger systems and amenable to further mathematical scrutiny.

PROGRAM FLOW GRAPH ANALYSIS

A system model can be represented by a flow chart or directed graph. The chart exhibits tasks as nodes and connecting arcs as the possible flow of control (decisions) between tasks or work unit modules. The transition between these nodes is a function of the probability of branching from one task to another. These transition probabilities are assured to be fixed and statistically independent. This leads to an analysis of a discrete Markov process. Ramamoorthy (1965) analyzed computer programs in this manner. The average execution time of each task is the product of the number of task iterations and the time to execute the task once.

Brzozowski and McCluskey (1963) applied signal flow graph techniques to the problem of characterizing sequential circuits. Kleir and Ramamoorthy (1971) used a program graph representation during translation in an optimizing microprogram compiler. Beizer (1970) measured the running time of programs using the Markov model with transition times measured by the number of instructions in a procedure.

In a file organization, all data base management functions consist of a search of the directory, a block search of the secondary storage data structure (if conditions warrant), and some action (retrieval, update). A macro-scopic view of the model in directed graph form is represented by Figure 1.

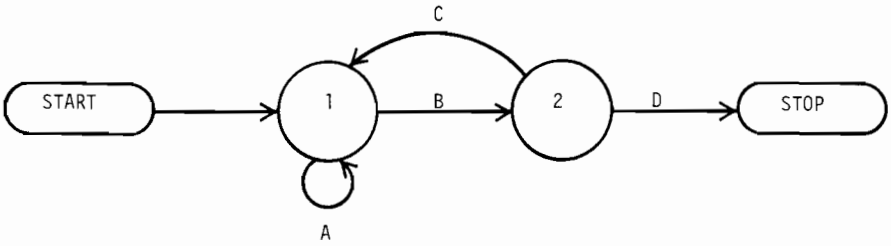


Figure 1
Flow Graph of General File Organization

The transitions between hierarchical levels (1 and 2) are represented by letters where A is the directory search iteration, B indicates a successful directory search, C implies a false drop has occurred and D is followed when the desired record is found. Level 1 is the directory search and level 2 is the secondary storage structure search.

Since the flow graph models a random process in which each transition is independent of the previous states and their probabilities, the flow graph is said to have the Markovian property. This is seen in Figure 2.

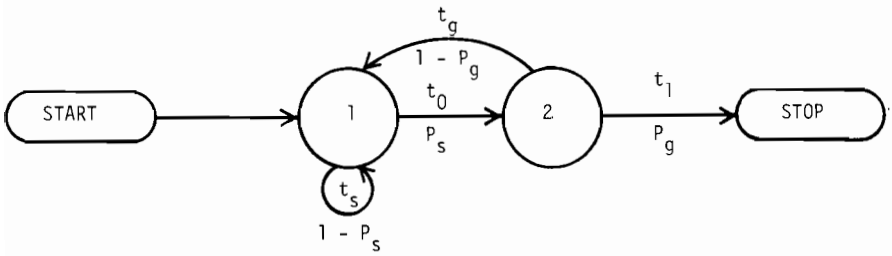


Figure 2
Markov Model of General File Organization

The Markov chain contains the variables

P_s - the probability of finding a match to the desired query in the directory,

P_g - the probability of finding the desired record in the file if the directory search is successful,

t_0 - time to search a secondary storage block,

t_1 - time to retrieve a secondary storage block,

t_g - time lost in a false drop search attempt,

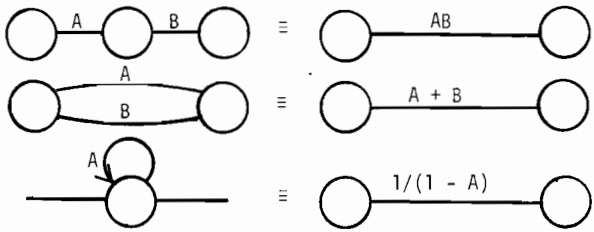
t_s - time to search a directory location.

The Markov model assumes the probabilities to be fixed and statistically independent. Therefore, the execution time for each level transition in the graph is

$$A = (1 - P_s)\, t_s$$
$$B = P_s\, t_0$$
$$C = (1 - P_g)\, t_g$$
$$D = P_g\, t_1.$$

REDUCTION TO A LINEAR EQUATION

Brzozowski and McCluskey (1963) proved that flow graphs can be reduced by simple algorithms to linear equations in the regular expression algebra. Basic equivalences are used to reduce flow graphs to regular expressions:



Thus, any state diagram can be reduced to a linear equation if a decision probability for transition between processes and the process execution times are known. Heistand (1964) defined the executive control program of an operating system monitor by reduction of a finite-state automaton. The regular expression algebra is defined by Kleene (1956).

The regular expression for the flow graph model of the general file organization in Figure 1 is

$$G = A^{*} \left[B + (BC)^{*} \right] D$$

where the operator, * , on a variable a is defined by $a^{*} = 1 / (1 - a)$. This expression leads to

$$G = (B + 1/(1 - BC))\, D / (1 - A)$$

Expanding,

$$G = (B - B^2C + 1)\ D\ /\ (1 - A - BC + ABC)$$

Generating functions used for algorithm analysis can be f[illegible]d in Knuth (1973).

The Z-TRANSFORM

A Z-transform serves as a discretized time delay operator. The application of the Z-transform represents a fixed time delay in making the transition from one event to another. The Z-transform is given by

$$Z\{f(t)\} = \sum_{t=0}^{\infty} f(t)Z^t$$

The Z-transform is introduced for the purpose of calculating the expected time delay (response time) for the entire model. Further explanations of Z-transforms and formalisms may be found in Gupta (1966) and Jury (1964). With the time delay operator, Z, the transitions became

$$A = (1 - P_s)\ Z^{t_s}$$
$$B = P_s Z^{t_0}$$
$$C = (1 - P_g)\ Z^{t_g}$$
$$D = P_g\ Z^{t_1}$$

DETERMINATION OF EXPECTED RESPONSE

If the generating function, G, (Z) represents the reduced and transformed expression, the expected response time of the model is

$$\left.\frac{\partial G(Z)}{\partial Z}\right|_{Z = 1.}$$

Substituting,

$$G(Z) = \frac{(P_s Z^{t_0} P_g Z^{t_1}) - (P_s^2 Z^{2t_0} (1 - P_g)\ A^{t_g} P_g Z^{t_1}}{1 - (1 - P_s)\ Z^{t_s} - (P_s Z^{t_0} (1 - P_g)\ Z^{t_g}) + (1 - P_s)\ Z^{t_s} P_s Z^{t_0} (1 - P_g) Z^{t_g}}$$

and,

$$G(Z) = \frac{P_s P_g Z^{t_0 + t_1} - P_g P_s^2 (1 - P_g)\ Z^{2t_0 + t_1 + t_g} + P_g Z^{t_1}}{1 - (1 - P_s)\ Z^{t_s} - P_s (1 - P_g)\ Z^{t_0 + t_g} + P_s (1 - P_s)(1 - P_g) Z^{t_0 + t_s + t_g}}$$

Differentiating by parts,

$$\left.\frac{\partial G(Z)}{\partial Z}\right|_{Z=1} = \frac{vdu - udv}{v^2}$$

with

$$u = P_s^{-1} P_g$$

$$V = P_s + (1 - P_s + P_s P_g)^{-1}$$

and

$$dv = t_0 P_s + (1 - P_s + P_s P_g)^{-2} (t_g P_s - t_g P_s P_g + t_0 P_s P_g)$$

$$du = t_1 P_s^{-1} P_g - t_s P_g P_s^{-1} + t_s P_g P_s^{-2} .$$

This ultimately reduces to

$$\left.\frac{\partial G(Z)}{\partial Z}\right|_{Z=1} = P_g \left[\left\{ t_0 + t_1 + (1 - P_g)(t_0 + t_g) - t_1 (1 - P_s^{-1} - P_g) + (1 - P_s^{-2} - P_g) t_s \right\} / \left\{ (1 - P_s + P_s P_g)^2 (1 - P_s + P_s P_g) \right\} \right]$$

which is the expected time to answer a simple query (referred to hereafter as E).

A cursory examination of E shows that E is directly proportional to P_g, the probability of finding the desired record in the file after a successful search of the directory. Also, P_s^{-1} and P_s^{-2} are significant for a small P_s.

Figure 3 shows the relationship between E, P_s, and P_g for constant t_0, t_1, and t_g. t_s is not significant because $t_s << t_0$, t_1, and t_g. E is at a maximum at $P_g = 1.0$ and $P = .05$. This is because the full retrieval cycle will be followed. Thus, more work is done as the probability of finding a successful directory match decreases (i.e. when P_s approaches 0) and the probability of a false drop decreases (when $P_g \rightarrow 1$).

Figure 4 illustrates the dependency of the response time on hardware parameters. It is seen that E is directly proportional to $t_0 = t_1 = t_g$. If the hardware configuration is altered to incur a timewise saving, the response time increases proportionately.

A cost/performance analysis for sensitivity can also be performed upon the expected response equation. An example can be found in Allen (1976).

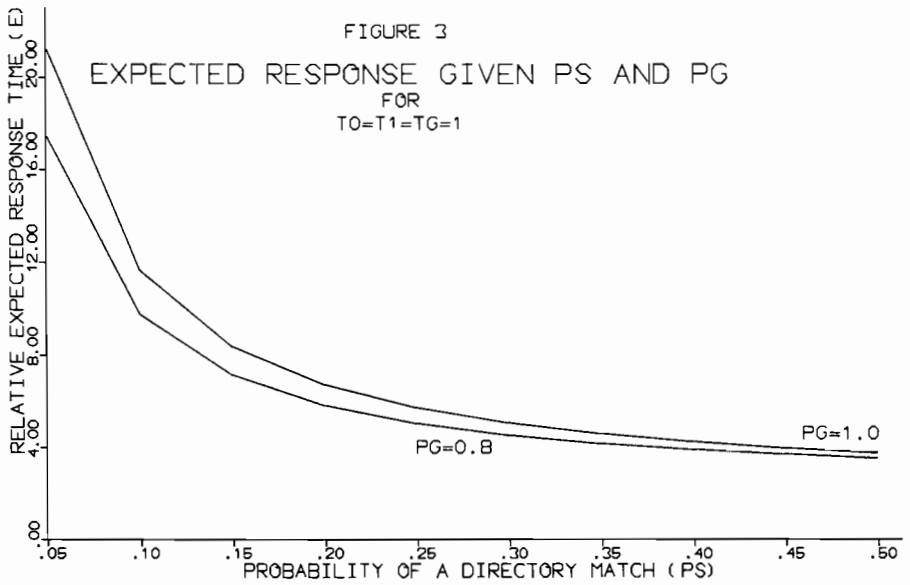


FIGURE 3
EXPECTED RESPONSE GIVEN PS AND PG
FOR
TO=T1=TG=1
RELATIVE EXPECTED RESPONSE TIME (E)
20.00
16.00
12.00
8.00
4.00
.00
PG=1.0
PG=0.8
.05
.10
.15
.20
.25
.30
.35
.40
.45
.50
PROBABILITY OF A DIRECTORY MATCH (PS)

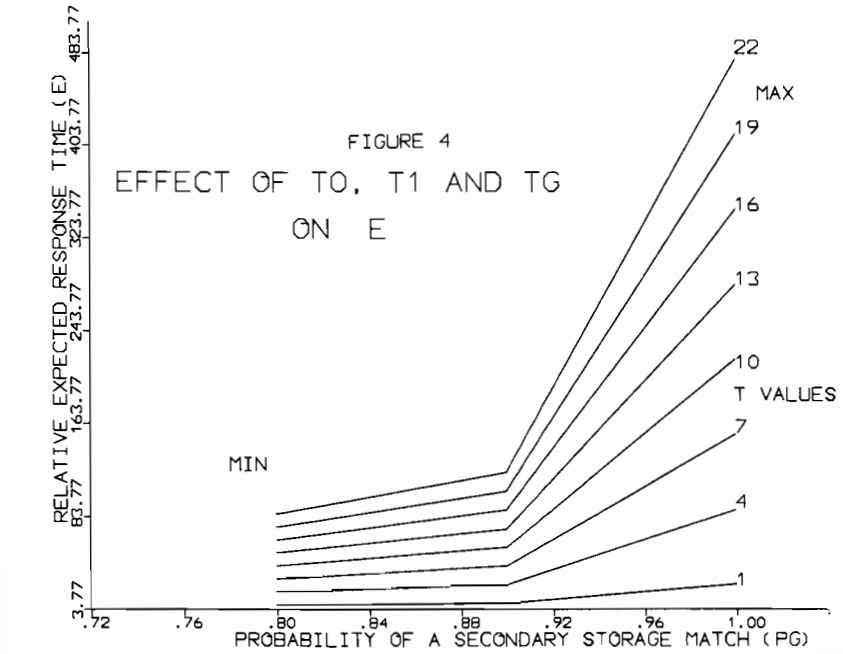


FIGURE 4
EFFECT OF TO, T1 AND TG
ON E
RELATIVE EXPECTED RESPONSE TIME (E)
483.77
403.77
323.77
243.77
163.77
83.77
3.77
22
MAX
19
16
13
10
T VALUES
7
4
1
MIN
.72
.76
.80
.84
.88
.92
.96
1.00
PROBABILITY OF A SECONDARY STORAGE MATCH (PG)

The mean response times are inadequate measures of performance if the variance of response times is great. To insure validation of the model presented, the second moment can be calculated by

$$VAR\ (G) = G''\ (1) + G'\ (1) - G'\ (1)^2.$$

APPLICATION TO LARGER SYSTEMS

If two functional modules of an information system are expressed as functions of G_1 and G_2, the mean response time of the combined modules is

$$E\ (G_1\ G_2) = E\ (G_1) + E\ (G_2).$$

The increasing complexity of large systems can be subdivided by task and described analytically through individual treatments. Then the components can be combined to model the entire system.

The parameters of the macro-systems can be varied over realistic ranges to provide the first-order approximations. The interpretation of the comparative statistics can aid management in decision-making processes. These decisions include system procurement alternatives, performance/cost resolution and functional objective levels.

CONCLUSION

The purpose of the throughput analysis is to enable the designer to determine approximately the parameters needed to achieve a given level of system performance relative to other alternatives. Algorithms were given for the design and analysis of systems which are applicable to MIS performance projection.

The analytical model derived was based upon known functional relationships. These relationships may not be clearly defined for some models. However, a model prototype describing the functional objectives of a proposed system should of necessity already exist or the planning operation is deficient.

Model complexity can create tedious mathematical computations during the expectation calculation. The effort is certainly no greater than programming synthetic programs for simulation. In addition, the parameters are more easily altered for analysis in an analytical model.

REFERENCES

|1| F.W. Allen: File Retrieval Through Information Filtering, Ph. D. dissertation. (University of Southwestern Louisiana, 1976).

|2| B. Beizer: Analytical Techniques for Statistical Evolution of Program Running Time, AFIPS Conference Proceedings. 37 (1970) 519-524.

|3| J.A. Brzozowski and E.J. McCluskey: Signal Flow Graph Techniques for Sequential Circuit State Diagrams, IEEE Tran. on Elec. Computers. 12 (April, 1963) 67-76.

|4| A.F. Cardenas: Evaluation and Selection of File Organizations - A Model and A System, C.A.C.M. 16 (September, 1973) 540-548.

|5| M.E. Drummond Jr.: A Perspective on System Performance Evaluation, IBM Systems Journal. 8 (1969) 252-263.

|6| S.P. Gosh and W.G. Tuel Jr.: A Design of an Experiment to Model Data Base System Performance, IEEE Tran. on Software Engr. SE-2 (June, 1976) 97-105.

|7| S.C. Gupta: Transform and State Variable Methods in Linear Systems, (New York: Wiley, 1966) 144-175.

|8| R.E. Heistand: An Executive System Implemented as a Finite-State Automaton. C.A.C.M. 7 (November, 1964) 699-677.

|9| E.I. Jury: Theory and Application of Z-Transform Method, (New York: Wiley, 1964).
|10| S.C. Kleene: Representation of Events in Nerve Nets and Finite Automata, ANN. of Math Studies. 34 (1956) 3-41.
|11| R.L. Kleir and C.V. Ramamoorthy: Optimization Strategies for Microprograms, IEEE Trans, Comp. C-20 (July, 1971) 783-794.
|12| D.E. Knuth: The Art of Computer Programming, 3 (Reading, Mass: Addison-Wesley, 1973).
|13| Henry C. Lucas Jr.: Performance Evaluation and Monitoring, Computer Surveys 3 (September, 1971) 79-91.
|14| C.V. Ramamoorthy: Discrete Markov Analysis of Computer Programs, Proc. A.C. M. 20th Nat'l Conf. (1965) 386-392.
|15| Liba Svobodava: Computer Performance Measurement and Evaluation Methods: Analysis and Applications, (New York: Elsevier, 1976).
|16| S.B. Yao: An Attribute Based Model for Database Access Cost Analysis, A.C. M. Trans on Database Sys. 2 (March, 1977) 45-67.

QUEUEING THEORY

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

SOLUTIONS OF FUNCTIONAL EQUATIONS ARISING IN THE ANALYSIS OF TWO SERVER QUEUEING MODELS

G. Fayolle
IRIA-LABORIA
78150 Le Chesnay
France

R. Iasnogorodski
Université d'Orléans
Orléans
France

Readers familiar with queueing theory know that many problems in this field are related to functional equations of special type. Broadly speaking, the unknown functions are the generating functions for a stationary distribution of the studied process. Our objective in this paper is to provide a method of solving such equations in the case of two dimensional random walks. Two particular problems are adressed but the procedures used are general.

1 - Problem A.

Two F.I.F.O parallel M/M/1 queues coupled according the following stategy : service times of the top-jobs in each queue have instantaneous service rates depending on the system state (see section A).
This problem is reduced to a Riemann-Hilbert problem and closed formulas are obtained for the generating function $F(x,y)$ (associated to the number of jobs in each queue) in terms of elliptic functions of third kind.

2 - Problem B.

We present here a solution to the problem of "joining the shorter of two unbounded F.I.F.O - M/M/1 queues". A job upon, arrival, is assigned to the shorter queue. If the queues have equal length, an arriving job joins queue i with probability π_i, i=1,2.
This problem has been studied only in the symmetric case (i.e. $\pi_1=\pi_2=\frac{1}{2}$ and equal servicing rates) :
- formerly by Kingman [10].
- recently by L. Flattó and MacKean [2] who improved Kingman's results and obtained the stationary probabilities.
The non symmetric case is more complex, because it requires a priori four unknown functions. We reduce it by computing a function satisfying a singular integral equation with a Noetherian operator.

PROBLEM A

A1. - Problem formulation and assumptions

Let us consider two parallel M/M/1 queues with infinite capacities under the following assumptions.

a) The arrivals form two independent Poisson processes with parameters λ_1, λ_2.

b) The service times are distributed exponentially with instantaneous service rates S_1 and S_2 depending on the system state in the following manner :

i) $S_1 = \mu_1$, $S_2 = \mu_2$ if both queues are busy

ii) $S_1 = \mu_1^*$ if queue 2 is empty

iii) $S_2 = \mu_2^*$ if queue 1 is empty

c) The service discipline is FIFO (first in - first out) in each queue.

Let $p_t(m,n)$ be the probability that, at time t, there are m jobs in queue 1 and n jobs in queue 2.

Note : For sake of brevity all proofs are omitted in the paper.

$p(m,n) = \lim_{t\to\infty} p_t(m,n)$ will be referred to as the stationary probability of the state (m,n). We study the behaviour of the system at the steady state by means of the generating function F(x,y) (see below).

From now on the terms "stability" or "ergodicity" will be used to mean "there exists a stationary distribution".

Using the Kolmogorov's forward equations for the p(m,n) and the generating function :

$$F(x,y) = \sum_{m=0}^{\infty} \sum_{n=0}^{\infty} p(m,n)\, x^m\, y^n$$

which is analytic with respect to x and y whenever $|x|, |y| < 1$, a straightforward but tedious computation yields :

$$T(x,y)\, F(x,y) = F(o,y)\, a(x,y) + F(x,o)\, b(x,y) + F(o,o)\, c(x,y) \tag{1.1}$$

where :

$$a(x,y) \overset{\text{def}}{=} \mu_1(1 - \tfrac{1}{x}) + q(1 - \tfrac{1}{y})$$

$$b(x,y) \overset{\text{def}}{=} \mu_2(1 - \tfrac{1}{y}) + p(1 - \tfrac{1}{x})$$

$$c(x,y) \overset{\text{def}}{=} p(\tfrac{1}{x} - 1) + q(\tfrac{1}{y} - 1)$$

$$T(x,y) \overset{\text{def}}{=} \lambda_1(1-x) + \mu_1(1 - \tfrac{1}{x}) + \lambda_2(1-y) + \mu_2(1 - \tfrac{1}{y})$$

$$p = \mu_1 - \mu_1^*$$

$$q = \mu_2 - \mu_2^*$$

Lemma 1.1. The existence of F(x,y) satisfying the functional equation (1.1) with $\sum_{m,n=0}^{\infty} |p(m,n)| < \infty$ (space L_1) is equivalent to stability. Moreover if F(x,y) exists, it is unique up to a constant multiplier which can be suitable chosen so that all coefficients in the power series expansion are positive and sum to 1. In that case F(x,y) is the generating function for a stationnary distribution.

Proof see Malyshev [5].

A glance at relation (1.1) does not give many information concerning F(x,y). However, a further investigation shows that the right side member vanishes whenever :

$$T(x,y) = 0, \quad \text{provided } |x|, |y| \leq 1.$$

The obtain additional relations between $F(x,0)$ and $F(0,y)$, it is necessary to examine carefully the algebraic curve C defined by :

$$R(x,y) = xy.T(x,y) = 0 \tag{1.2}$$

in the whole complex plane, what is done in the next section.

A2. - $R(x,y) = 0$

$R(x,y)$ is a polynomial of third degree w.r.t. x and y and of second degree w.r.t. each variable x or y.

The curve C has genus 1 and can be identified with the Riemann surface C over either of the extended x or y planes (more precisely, the algebraic extension of the field of rational functions of x, as defined by $R(x,y) = 0$).

These assertions are well known (see for example Fuchs [3]) and will not be discussed further.

For curves of genus 1, the introduction of a uniformizing parameter requires elliptic functions and reduces the tractability of the computation if we try to construct the analytic continuation of the functions $F(x,0)$ and $F(0,y)$ (for an example concerning a curve of genus 0, see Flatto - MacKean [2]).

Solving $R(x,y) = 0$ for y, we have :

$$y(x) = \frac{\lambda_2+\lambda_2+A(x) \pm \sqrt{(\lambda_2+\mu_2+A(x))^2 - 4\lambda_2\mu_2}}{2\lambda_2} \tag{2.1}$$

where :

$$A(x) = \lambda_1(1-x) + \mu_1\left(1 - \frac{1}{x}\right) \tag{2.2}$$

We obtain two distinct branches which give a two sheeted covering over the x plane.

Lemma 2.1. The algebraic function $y(x)$ defined by $R(x,y)$ has four real branch points x_1, x_2, x_3, x_4 with $0 < x_1 < x_2 < 1 < x_3 < x_4$.

The lemma 2.1 is obviously valid for the branch points y_1, y_2, y_3, y_4 of the function $x(y)$.

Lemma 2.2. The equation $R(x,y) = 0$ has one root $y(x) = h(x)$ which is an analytic algebraic function of x in the whole complex plane cut along the two segments $[x_1x_2]$ and $[x_3x_4]$.

Moreover $|h(x)| \leq 1$ if $|x| = 1$:

$$|h(x)| \leq \sqrt{\frac{\mu_2}{\lambda_2}} \quad \forall\ x.$$

Similar propositions apply to $x(y)$: i.e. there exists $k(y)$ such that

$$(k(y),y) = 0$$
$$|k(y)| \leq 1 \text{ if } |y| = 1$$
$$|k(y)| \leq \sqrt{\frac{\mu_1}{\lambda_1}} \quad \forall\ y.$$

Lemma 2.3.

$$k(h(x)) = \begin{cases} x & \text{if } |x| \leq \sqrt{\frac{\mu_1}{\lambda_1}} \\ \frac{\mu_1}{\lambda_1 x} & \text{if } |x| > \sqrt{\frac{\mu_1}{\lambda_1}} \end{cases}$$

(2.3)

$$h(k(y)) = \begin{cases} y & \text{if } |y| \leq \sqrt{\frac{\mu_2}{\lambda_2}} \\ \frac{\mu_2}{\lambda_2 y} & \text{if } |y| > \sqrt{\frac{\mu_2}{\lambda_2}} \end{cases}$$

A3. - The analytic continuation of F(x,0) and F(0,y)

Convention : To avoid repetitions, we introduce the notations :

$$C(R) = \{z | \text{modulus } (z) = R\}$$
$$B(R) = \{z | \text{modulus } (z) < R\}$$
$$\overline{B(R)} = \{z | \text{modulus } (R) > R\}$$

In section II, we have defined two curves :

$$H = \{h(z)/|z| = 1\}$$
$$K = \{k(z)/|z| = 1\}.$$

which are simple, closed and lie inside C(1). Similarly, let H' and K' be the curves obtained from H and K under the mappings (automorphisms) $z \to \frac{\mu_2}{\lambda_2 z}$ and $z \to \frac{\mu_1}{\lambda_1 z}$ respectively, i.e. :

$$H' = \frac{\mu_2}{\lambda_2 H}$$

$$K' = \frac{\mu_1}{\lambda_1 K}.$$

H' and K' are also simple, closed and lie outside C(1).
(The point z=1 on the positive real axis belongs to H' (resp. K') iff $\frac{\mu_2}{\lambda_2} < 1$ (resp. $\frac{\mu_1}{\lambda_1} < 1$) as shown in section II).

We shall use throughout this paper the functions h(x) and k(y) introduced in II.

Theorem 3.1 : 1) F(x,0) (resp. F(0,y)) can be continued as a meromorphic function to the whole complex plane cut along the real axis from x_3 to x_4 (resp. from y_3 to y_4).

2) The poles of F(x,0) (resp. F(0,y)) are the zeros (if any) of $\mu_2(1 - \frac{1}{h(x)}) + p(1 - \frac{1}{x})$ (resp. $\mu_1(1 - \frac{1}{k(y)} + q(1 - \frac{1}{y}))$ outside C(1).

To prove theorem 3.1 we need the :

Lemma 3.1. All the couples (x,y) which are solutions of the system

$$(3.1)\qquad \begin{cases} R(x,y) = 0 \\ |x| \leq 1 \\ |y| \leq 1 \end{cases}$$

have the form $(x,h(x))$ or $(k(y),y)$ where $x \in R_x$ and $y \in R_y$.

Applying the notation of section A1., system (3.1) entails :

$$(3.2)\qquad \boxed{F(0,y)\, a(x,y) + F(x,0)\, b(x,y) + F(o,o)\, c(x,y) = 0}$$

A4. - A simplification of the relation (3.3)

We try to reduce (3.3) to an homogeneous equation (i.e. without right member).
An elementary computation shows that when :

$pq - \mu_1\mu_2 \neq 0$, the transformation (translation).

$$(4.0)\qquad \begin{cases} F(0,y) = \dfrac{p(q-\mu_2)}{pq-\mu_1\mu_2} F(0,0) + H(y) \\ F(x,0) = \dfrac{q(p-\mu_1)}{pq-\mu_1\mu_2} F(0,0) + G(x) \end{cases}$$

yields the system :

$$(4.1)\qquad \begin{cases} R(x,y) = 0, \quad |x|, \ |y| \leq 1 \\ H(y)\, a(x,y) + G(x)\, b(x,y) = 0. \end{cases}$$

Moreover,

$$(4.2)\qquad \begin{cases} H(0) = \dfrac{\mu_2\mu_1^* F(0,0)}{\mu_1\mu_2 - pq} \\ G(0) = \dfrac{\mu_1\mu_2^* F(0,0)}{\mu_1\mu_2 - pq} \end{cases}$$

Section A3. gives the analytic continuations of $G(x)$ and $H(y)$.

When $pq=\mu_1\mu_2$, the preceding transformation is not valid. However, remembering that :

$$p = \mu_1 - \mu_1^* \quad \text{and} \quad q = \mu_2 - \mu_2^*,$$

$pq - \mu_1\mu_2 = 0$ can be rewritten as :

$$\frac{\mu_1}{\mu_1^*} + \frac{\mu_2}{\mu_2^*} = 1 ,$$

or

$$(4.3)\qquad \begin{cases} \mu_1 = \xi\, \mu_1^* \\ \mu_2 = (1-\xi)\mu_2^* \end{cases} \qquad 0 \leq \xi \leq 1, \text{ provided that } \mu_1^*, \mu_2^* \neq 0$$

If $\mu_1^* = \mu_2^* = 0$, (4.1) yields the system :

$$\begin{cases} R(x,y) = 0, \quad |x|, \ |y| \le 1 \\ F(0,x) + F(y,0) = F(0,0), \ (x,y) \neq (1.1) \ , \end{cases}$$

which clearly has no admissible solution due to the fact that $F(x,0)+F(0,y) > F(0,0)$.

Using (4.3), (4.1) takes the following form :

$$(4.4) \qquad \begin{cases} R(x,y) = 0, \quad |x| \le 1, \quad |y| \le 1 \\ [\mu_1^*(1 - \frac{1}{x}) - \mu_2^*(1 - \frac{1}{y})][\xi F(0,y)-(1-\xi)F(x,0)] = \\ \qquad = [\mu_1^*(\frac{1}{x} - 1)(1-\xi) + \mu_2^*(\frac{1}{y} - 1)\xi] F(0,0). \end{cases}$$

A5. - The roots of $b^1(x) = p(1 - \frac{1}{x}) + \mu_2(1 - \frac{1}{h(x)}) = 0$ inside $C(\sqrt{\frac{\mu_1}{\lambda_1}})$

Following the remark in section A3., it will be assumed from now on $\mu_1 > \lambda_1$.
Section III reveals that a stepping stone towards the solutions of the original problem is the study of the roots of the equation $b^1(x)=0$. More precisely, we require only the roots inside $C(\sqrt{\frac{\mu_1}{\lambda_1}})$ for reasons given in sections VI and VII. It seems convenient to introduce the following notations (valid until the end of the paper).

$$(5.1) \qquad \begin{cases} b^1(x) = p(1 - \frac{1}{x}) + \mu_2(1 - \frac{1}{h(x)}) \\ a^1(x) = q(1 - \frac{1}{h(x)}) + \mu_1(1 - \frac{1}{x}) \\ b^2(y) = p(1 - \frac{1}{k(y)}) + \mu_2(1 - \frac{1}{y}) \\ a^2(y) = q(1 - \frac{1}{y}) + \mu_1(1 - \frac{1}{k(y)}) \end{cases}$$

Up to change of the parameters, the conclusions drawn for $b^1(x)=0$ will hold for $a^2(y)=0$.
The term "root" or "zero" will always design a number different from 1.

Lemma 5.1. Excluding the trivial root x=1, $b^1(x)=0$ has at most two roots inside $C(\sqrt{\frac{\mu_1}{\lambda_1}})$. Moreover, these roots (if any) are real , positive and belong to the set $[0,x_1] \cup [x_2,\sqrt{\frac{\mu_1}{\lambda_1}})[$

A6. - The case $pq=\mu_1\mu_2$: determination of F(x,0) and F(0,y) by solving a Dirichlet problem for a circle.

From now on, let $\mathfrak{J}_m(z)$ and $\mathfrak{R}_e(z)$ denote respectively the imaginary and the real part of the complex number z.
From section I, we know that $\mathfrak{J}_m F(0,y) = 0$ for $y \in [y_1 y_2]$, since the power series expansion of F(o,y) has positive coefficients.
Hence, by using system (4.4) and section III, it follows :

$$(6.1) \qquad \mathfrak{J}_m(1-\xi)F(x,0) = \mathfrak{J}_m \frac{\mu_2^*(\frac{1}{h(x)} - 1) F(0,0)}{-\mu_1^*(1 - \frac{1}{x}) + \mu_2^*(1 - \frac{1}{h(x)})} \quad \text{for } |x| = \sqrt{\frac{\mu_1}{\lambda_1}}$$

Here, $b^1(x) = -\mu_1^*(1 - \frac{1}{x}) + \mu_2^*(1 - \frac{1}{h(x)})$.

- if $b^1(x)$ has no zero in $B(\sqrt{\frac{\mu_1}{\lambda_1}})$, we reduce the problem to that of finding a function $F(x,o)$ analytic in $B(\sqrt{\frac{\mu_1}{\lambda_1}})$ continuous on $C(\sqrt{\frac{\mu_1}{\lambda_1}})$ and satisfying the boundary condition (6.1). This is a particular case of a Dirichlet problem for a circle.
- if $b^1(x)$ has a zero in $B(\sqrt{\frac{\mu_1}{\lambda_1}})$ say x_o, we have still a Dirichlet problem for the function $(x-x_o)\ F(x,o)$.

Provided that $b^1(x)$ has no root in $B(\sqrt{\frac{\mu_1}{\lambda_1}})$, $F(x,0)$ is determined from (6.1), in $B(\sqrt{\frac{\mu_1}{\lambda_1}})$, up to a constant, by Schwarz's formula Muskhelishvili [6] Section 41.

$$F(\frac{\mu_1}{\lambda_1}z,0) = \frac{i}{2\pi}\int_{-\pi}^{\pi} u(\rho)\ \frac{e^{i\rho}+z}{e^{i\rho}-z}\ d\rho + D, \qquad |z| < 1 \tag{6.2}$$

where D is a real constant :

and $$u(\rho) = \frac{1}{1-\xi}\,\mathfrak{I}_m\ \frac{\mu_2^*(1-\frac{1}{h(x)})\ F(0,0)}{\mu_1^*(1-\frac{1}{x})-\mu_2(1-\frac{1}{h(\bar{x})})}$$

With $x = \sqrt{\frac{\mu_1}{\lambda_1}}\ e^{i\rho}$.

$$u(\rho) = \frac{-\lambda_1 \sin\rho\ H(\rho)}{[\rho_2^*(\mu_1^*-\mu_2^*)H^2(\rho)+(\mu_2^*-\mu_1^*+\lambda_1+\lambda_2)H(\rho)-\mu_2^*](1-\xi)} \tag{6.3}$$

where : $H(\rho) = h(\sqrt{\frac{\mu_1}{\lambda_1}}\ e^{i\rho})$

$$= \frac{\lambda_2+\mu_2+\alpha - \sqrt{[(\sqrt{\lambda_2}+\sqrt{\mu_2})^2+\alpha][(\sqrt{\lambda_2}-\sqrt{\mu_2})^2+\alpha]}}{2\lambda_2}$$

and $\alpha = \lambda_1 + \mu_1 - 2\sqrt{\lambda_1\ \mu_1}\ \cos\rho$.

Note that $H(\rho)$ is real. From (6.3), we deduce easily that $u(\rho)$ is an odd function of ρ. This implies :

$$\boxed{F(\sqrt{\frac{\mu_1}{\lambda_1}}z,0) = \frac{1}{\pi}\int_o^{\pi} \frac{z \sin\rho\ u(\rho)d\rho}{1+z^2-2z\cos\rho} + F(0,0) \qquad |z| < 1} \tag{6.4}$$

The expression of $F(\sqrt{\frac{\mu_1}{\lambda_1}}z,0)$ in terms of elliptic functions of the third kind can be derived. [Appendix omitted]

A similar formula holds for $F(0,\sqrt{\frac{\mu_2}{\lambda_2}}z)$ provided that the system is ergodic which is equivalent to $a^2(y) \neq 0$ in $C(1)$ (whatever the position of $\frac{\mu_2}{\lambda_2}$ w.r.t. 1 may be) or :

$$\boxed{1 - \rho_1^* - \rho_2^* > 0} \tag{6.5}$$

A7. - The case $pq \neq \mu_1\mu_2$: determination of $F(x,o)$ and $F(o,y)$ by solving an homogeneous Riemann-Hilbert problem for a circle.

In that section we use intensively the procedure given in Muskhelishvili [6], chapter 5, sections 39-40.

Theorem 7.1. Our problem is a particular case of a famous general problem due to Riemann and first studied by Hilbert.
This problem is a follows :

Let S^+ be a finite or infinite region, bounded by a single simple contour L. It is required : to find a function $\Omega(z) = u+iv$, holomorphic in S^+ and continuous in $S^+ + L$, satisfying the boundary condition :

$$\mathcal{R}e[U(z) \cdot \Omega(z)] = V(z) \text{ on } L, \tag{7.1}$$

where U(z), V(z) are continuous functions given on L. The homogeneous problem is obtained by putting : $V(z) \equiv Q$ in (7.1).

Demonstration.

Lemma 7.1.

$$G(z) = \frac{[z-k(\alpha_2)]^{i_1}\,[z-k(\beta_2)]^{i_2}\,\tilde{G}(z)}{[z-\gamma_1]^{i_3}}, \qquad |z| < \sqrt{\frac{\mu_1}{\lambda_1}} \tag{7.2}$$

where i_j, j=1,2,3 takes the values 0 or 1 and :

i) γ_1 is the eventual zero of $b^1(x)$ in $[1,\sqrt{\frac{\mu_1}{\lambda_1}}]$

ii) α_2, β_2 are the eventual zeros of $a^2(y)$ in $[0,y_1] \cup [y_2,1]$;

iii) $\tilde{G}(z)$ is analytic in $B(\sqrt{\frac{\mu_1}{\lambda_1}})$.

Upon setting :

$$U(z) = \frac{b^1(z)}{a^1(z)} \cdot \frac{[z-k(\alpha_2)]^{i_1}[z-k(\beta_2)]^{i_2}}{(z-\gamma_1)^{i_3}} \tag{7.3}$$

It follows :

$$\mathcal{R}e[i\,U(z)\,\tilde{G}(z)] = 0 \quad \text{for } |z| = \sqrt{\frac{\mu_1}{\lambda_1}} \tag{7.4}$$

From [6] formula (40-10) - § 40, $\tilde{G}(z)$ is given by :

$$\tilde{G}(z) = D\,e^{\Gamma(z)} \quad \text{for } |x| < \sqrt{\frac{\mu_1}{\lambda_1}} \tag{7.5}$$

where D is a constant, non zero, and :

$$\Gamma(z) = \frac{1}{2i\pi} \int_{C(\sqrt{\frac{\mu_1}{\lambda_1}})} \frac{\log[t^{-\chi}J(t)]dt}{t-z} \tag{7.6}$$

where :

$$J(t) = \frac{\overline{iU(t)}}{iU(t)} \text{ and :}$$

$$\chi = \frac{1}{2i\pi} \log[\frac{\bar{U}(t)}{U(t)}] \text{ or :}$$

$$\chi = -\frac{1}{\pi}[\arg U(t)]_{C(\sqrt{\frac{\mu_1}{\lambda_1}})}, \text{ denoting by arg (z) the function "argument of z".} \tag{7.7}$$

Lemma 7.2.

$$\chi = 2[N_p.\mathrm{sgn}(\mu_2-\lambda_2)-N_z] + \frac{1}{\pi}\arg\frac{b^1(x)}{a^1(x)}\quad \overrightarrow{\leftarrow}_{x_1x_2} \tag{7.8}$$

(denoting by sgn(x) the function "sign of x" with sgn(0)=0) where :

* N_p is the number of zeros of $a^2(y)$ on $[1,\sqrt{\frac{\mu_2}{\lambda_2}}]$
* N_z is the number of zeros of $b^1(x)$ on $[0,x_1] \cup [x_2,1]$
* $\arg\left[\frac{b^1(x)}{a^1(x)}\right]_{\overrightarrow{\leftarrow}_{x_1x_2}}$ is the variation of the argument of $\frac{b^1(x)}{a^1(x)}$ along the "contour" $[x_1,x_2]$, starting from x_1 above $[x_1x_2]$, going to x_2 and coming back to x_1 below $[x_1x_2]$.

Lemma 7.3.

$$\frac{1}{\pi}\arg\frac{b^1(x)}{a^1(x)}\quad \overrightarrow{\leftarrow}_{x_1x_2} = \mathrm{sgn}(\mu_1\mu_2-pq)\left[\mathrm{sgn}\frac{b^1(x_1)}{a^1(x_1)} - \mathrm{sgn}\frac{b^1(x_2)}{a^1(x_2)}\right] \tag{7.9}$$

Lemma 7.4.

1) For $\chi \le -2$, the homogeneous Riemann-Hilbert probem has no solutions different from zero.

2) For $\chi \ge 0$, the homogeneous Riemann-Hilbert problem has exactly $\chi+1$ linearly independent solutions; the general solution is given by :

$$\Theta(z) = \tilde{G}(z)\,(c_0z^{\chi}+c_1z^{\chi-1} \ldots + c_{\chi})$$

where $c_0, c_1, \ldots, c_{\chi}$ are constants subject to $c_{\chi} = \bar{c}_{\chi-k}$, $k=\alpha,1,\ldots,\chi$ but otherwise arbitrary.

Demonstration. See Muskhelivshvili [6] S 40-p. 100.

Theorem 7.2. 1) the homogeneous Hilbert problem satisfying the boundary condition (7.4) has, at most, one solution. In other words, $\chi \le 0$.

2) The system is ergodic iff $\chi = 0$, which is equivalent to :

$$\left\{\begin{array}{l} \left.\frac{db^1(x)}{dx}\right|_{x=1} < 0 \iff \mu_1^* > \frac{\mu_2\lambda_1-\mu_1\lambda_2}{\mu_2-\lambda_2} \\ \left.\frac{da^2(y)}{dy}\right|_{y=1} < 0 \iff \mu_2^* > \frac{\mu_1\lambda_2-\mu_2\lambda_1}{\mu_1-\lambda_1} \end{array}\right\} \quad \text{if } \mu_2 \ge \lambda_2 \tag{7.10}$$

$$\left.\frac{da^2(y)}{dy}\right|_{y=1} < 0 \iff \mu_2^* > \frac{\mu_1\lambda_2-\mu_2\lambda_1}{\mu_1-\lambda_1} \quad \text{if } \mu_2 \le \lambda_2 \tag{7.11}$$

Theorem 7.3. Assuming (7.11) or (7.10), F(z,0) is given by :

$$\frac{F(z,0)}{F(0,0)} = \frac{q\mu_1^*}{\mu_1\mu_2-pq} + \frac{\mu_1\mu_2^*}{\mu_1\mu_2-pq}\,\frac{G(z)}{G(0)}, \quad |z| < \sqrt{\frac{\mu_1}{\lambda_1}} \tag{7.12}$$

where $\star\ F(0,0) = \left[\frac{\mu_2\lambda_1 - \mu_1\lambda_2 + \mu_2^*(\mu_1 - \lambda_1)}{\mu_1\mu_2^*}\right] \cdot \frac{G(0)}{G(1)}$

$\star$ G(z) is derived from (7.2) and (7.3) using :

$$\Gamma(z) = \frac{1}{2\pi} \int_{C(\sqrt{\frac{\mu_1}{\lambda_1}})} \frac{\Delta(t)dt}{t-z} \quad \text{and} \quad \Delta(t) = \arg\left[\frac{-\bar{U}(t)}{U(t)}\right]$$

A8. - Application to general two dimensional random walks [t.d.r.w]

The method used in section A6 and A7 can be applied in a more general context. Let us assume that a t.d.r.w. has a stationary distribution with a generating function $F(x,y) = \sum_{i,j} p_{ij} x^i y^j$ satisfying $\sum |p_{ij}| < \infty$ (space L_1) and the following functional equation $T(x,y).F(x,y) = A(x,y).F(x,0) + B(x,y).F(0,y) + C(x,y)$ where A(x,y), B(x,y) and C(x,y) are known and continuous. Then the determination of F(x,y) reduces to the solution of a Riemann-Hilbert problem whenever the following four conditions are met (the notation is the same as before) :

1) The continuous function k(y) (resp. h(x)) is such that $|k(y)| \leq 1$ (resp. $|h(x)| \leq 1$) if $|y| = 1$ (resp. $|x| = 1$.

2) k(y) (resp. h(x)) has two branch points inside C(1), y_1 and y_2 (resp. x_1 and x_2) which are the ends of a "cut" $[y_1,y_2]$(resp. $[x_1x_2]$).

3) The cut $[y_1y_2]$ (resp. $[x_1x_2]$) is mapped onto a simple closed curve C_k (resp. C_h) under the mapping $y \to k(y)$ (resp. $x \to h(x)$).

4) Moreover, the regions inside C(1) and C_k(resp. C_h) must have a non-empty intersection D_k(resp. D_h) such that $|k(y)| \leq 1$ (resp. $|h(x)| \leq 1$) for $y \in D_k$ (resp. $x \in D_h$).

Up to a conformal transformation, condition 3) says that it is possible to reduce the general case to that of a circular region. Specifically, the four preceding conditions hold in the problem of t.d.r.w. studied by Malyshev [5]. It is worthwhile to note the possibility of solving the Hilbert problem for arcs by using the arc $[x_3x_4]$ and the analytic continuation of F(x,0) to the whole complex plan (see Section III).

When the genus of R(x,y) is greater than 1, some of the conditions 1 to 4 may be not satisfied. Then, the method is still valid, (an example will be given in a future paper) but the computations are more complex.

PROBLEM B "JOINING THE SHORTER QUEUE"

B1. - Problem Formulation and Assumptions

Let us consider two parallel M/M/1 queues with infinite capacities and exponential service time distributions with means $\frac{1}{\alpha}$ for queue 1 and $\frac{1}{\beta}$ for queue 2 under the following assumptions.

a) the arrivals form a Poisson stream with mean λ.

b) a customer, upon arrival, is assigned to the shorter queue.

c) if the queues have equal length, the arriving customer joins queue i with probability π_i, i=1,2 and $\pi_1+\pi_2=1$.

To study the behaviour of this system at the steady state, we need the following functions : (analytic w.r.t x and y whenever $|x|, |y| < 1$) :

$$
(1.1)\quad \left\{
\begin{array}{l}
F_1(x,y) = \sum_{i,j=0}^{\infty} p(i,i+j)x^i y^j \\
F_2(x,y) = \sum_{i,j=0}^{\infty} p(i+j,i)x^i y^j \\
P_1(x) = \sum_{i=0}^{\infty} p(i+1,1)\, x^i \; ; \qquad A_1(x) = (\alpha+\lambda x)\, P_2(x) + \beta F(0,0) \\
P_2(x) = \sum_{i=0}^{\infty} p(i,i+1)\, x^i \; ; \qquad A_2(x) = (\beta+\lambda x)\, P_1(x) + \alpha F(0,0) \\
Q(x) = F_1(x,o) = F_2(x,o) = \sum_{i=0}^{\infty} p(i,i)x^i \; ; \quad G_i(y) = F_i(o,y)\ , \quad i=1,2 \\
T_1(x,y) = \lambda(1 - \frac{x}{y}) + \alpha(1 - \frac{y}{x}) + \beta(1 - \frac{1}{y}) \\
T_2(x,y) = \lambda(1 - \frac{x}{y}) + \beta(1 - \frac{y}{x}) + \alpha(1 - \frac{1}{y}) \\
S = \lambda + \alpha + \beta
\end{array}
\right.
$$

The Komogorov's forward equations for the p(m,n) yield

$$
(1.2)\quad \left\{
\begin{array}{l}
F_1(x,y)\, T_1(x,y) = \alpha(1 - \frac{y}{x})\, G_1(y) + (\frac{\lambda\pi_2 y^2 - \lambda x - \beta}{y})\, Q(x) + A_1(x) \\
F_2(x,y)\, T_2(x,y) = \beta(1 - \frac{y}{x})\, G_2(y) + (\frac{\lambda\pi_1 y^2 - \lambda x - \alpha}{y})\, Q(x) + A_2(x) \\
S\, Q(x) = A_1(x) + A_2(x)
\end{array}
\right.
$$

Lemma 1.1. The existence of $F_1(x,y)$ and $F_2(x,y)$ with $\sum_{m,n=0}^{\infty} |p(m,n)| < \infty$ (space L_1) is equivalent to stability. Moreover, if both $F_1(x,y)$ and $F_2(x,y)$ exist, they are unique up to a constant multiplier.

System (1.2) gives the minimum number (four) of unknown functions of one complex variable sufficient to describe the steady state. This number can be reduced at once, using the fact that, whenever $T_i(x,y)=0$, i=1,2 provided $|x|, |y| < 1$, the corresponding right side members in (1.2) vanish.
It remains nevertheless two unknown functions, for instance $G_i(y)$, i=1,2.

As in problem A, it is necessary to examine the algebraic curves $\mathcal{C}_i$, defined by

$$
(1.3)\qquad R_i(x,y) = xy.T_i(x,y),\ i=1,2\ .
$$

B2. - $R_1(x,y) = 0$

$R_1(x,y)$ is a polynômial of second degree w.r.t (x,y) and w.r.t. each variable x or y. The curve $\mathcal{C}_1$ has genus 0 and can be identified with the Riemann surface $\mathcal{C}_1$ over either of the extended x or y planes.

Solving $R_1(x,y)=0$ for y, we have :

$$(2.1) \qquad y(x) = \frac{sx \pm \sqrt{x[x(s^2-4\alpha\lambda) - 4\alpha\beta]}}{2\alpha}$$

Lemma 2.1. The algebraic function $y(x)$ defined by (2.1) has two real branch points 0, x^*, with $x^* < 1$.

Lemma 2.2. $R_1(x,y)=0$ has one root $Y_1(x)$ which is analytic in the whole complex plane cut along the segment $[0,x^*]$
Moreover $|Y_1(x)| \leq 1$ if $|x| = 1$.

The other root will be denoted by $Y_2(x)$.

Similar propositions apply to $x(y)$. (The two branch points are then denoted by

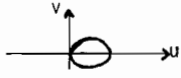
$$y_1^*,\ y_2^* \text{ with } 0 < y_1^* < y_2^* < 1.$$

Lemma 2.3.

i) $[0,x^*] \xrightarrow{Y_1=\bar{Y}_2} \varepsilon$

$[y_1^*,y_2^*] \xrightarrow{X_1=\bar{X}_2} \phi$

ε is the ellipse



ϕ is the ellipse

From now on, ε^i, ϕ^i [resp. ε^e, ϕ^e] will betoken the regions inside [resp. outside] the ellipses ε,ϕ.

ii) $\varepsilon - [0,x^*] \underset{Y_1}{\overset{X_1}{\rightleftarrows}} \phi - [y_1^*,y_2^*]$

iii) The following relations hold :

$$Y_1(X_2(z)) = z \quad \forall\ z$$

$$Y_2(X_1(z)) \begin{cases} = z & \text{if } z \in \varepsilon^e \cup \varepsilon \\ \neq z & \text{if } z \in \varepsilon^i \end{cases}$$

$$Y_1(X_1(z)) \begin{cases} = z & \text{if } z \in \varepsilon^i \cup \varepsilon \\ \neq z & \text{if } z \in \varepsilon^e \end{cases}$$

$$|Y_2(X_2(z))| > z \quad \forall\ z$$

Exchanging Y_i and X_i on the one hand, ε and ϕ on the other hand, 4 additional relations could be written

- Obviously, all conclusions drawn for $R_1(x,y)=0$ are valid for $R_2(x,y)=0$ up to an exchange of the parameters α and β . We need now additional notations to distinguish which equation $R_i(x,y)=0$ will be taken into account in such or such proof.

For this reason, we will just add the subscript (or superscript at own convenience). "(α,β)" [resp. "$(\beta\alpha)$"] , with or without brackets, in the quantities relating to $R_1(x,y)$ [resp. $R_2(x,y)$] .

B3. The analytic continuation of the functions $Q(z)$, $G_i(z)$, $A_i(z)$, $i=1,2$.

Theorem 3.1. All the functions of system (1.2) can be continued as meromorphic functions to the whole complex plane.

We just give here the sketch of the proof :

Upon defining recursively $\mathcal{D}_n = \inf[(X_2(Y_2))^{\alpha\beta}(\mathcal{D}_{n-1}),(X_2(Y_2))^{\beta\alpha}(\mathcal{D}_{n-1})]$ with $\mathcal{D}_o = \mathcal{D} = \{z/, |z|=1\}$ (the unit disk), one can show :

i) $\mathcal{D}_{n+1} \supset \mathcal{D}_n$, $\forall$ n

ii) $Q(x)$ is meromorphic in $\mathcal{D}_n$, $\forall$ n

iii) $\lim_{n\to\infty} \mathcal{D}_n$ covers the whole complex plane

The proof of ii) needs the following equation, which can be obtained from system (1.2)

$$(3.1) \qquad Q(x) = \frac{1}{\Delta(x)} \{\alpha(1 - \frac{Y_1^{\alpha\beta}(x)}{x}) G_1(Y_1^{\alpha\beta}(x)) + \beta(1 - \frac{Y_1^{\beta\alpha}(x)}{x}) G_2(Y_1^{\beta\alpha}(x))\}$$

with

$$(3.2) \qquad \Delta(x) \overset{def}{=} S - (\frac{\alpha}{x} + \lambda\pi_2) Y_1^{\alpha\beta}(x) - (\frac{\beta}{x} + \lambda\pi_1) Y_1^{\beta\alpha}(x)$$

This last quantity $\Delta(x)$, plays an important role. More precisely it is a "generator" for the poles of the various functions which occur in the problem.

B4. - A functional equation for $G_1(y)$.

From the preceding section, it is easy to prove the validity of the following equation in the complex plane :

$$(4.1) \qquad Q(x) = \frac{1}{W^{\alpha\beta}(x)} [G_1(Y_2^{\alpha\beta}(x))\alpha(1 - \frac{Y_2^{\alpha\beta}(x)}{x}) - G_1(Y_1^{\alpha\beta}(x)) \alpha(1 - \frac{Y_1^{\alpha\beta}(x)}{x})]$$

where

$$(4.2) \qquad W^{\alpha\beta}(x) = [Y_1^{\alpha\beta}(x) - Y_2^{\alpha\beta}(x)] [\lambda\pi_2 + \frac{\alpha}{x}],$$

A similar equation can be obtained for $A_1(x)$.

Upon combining these two equations and using the fact that, on the cut $[y_1^*,y_2^*]^{\beta\alpha}$, $G_2(y)$ must be continuous (then it will be analytic) [as in problem A] , we get

$$(4.3) \qquad 0 = \mathcal{J}_m \frac{1}{1 - \frac{y}{X_1^{\beta\alpha}(y)} W^{\alpha\beta}(X_1^{\beta\alpha}(y))} \{U_1(y)G_1[Y_2^{\alpha\beta}.X_1^{\beta\alpha}(y)] - V_1(y)G_1[Y_1^{\alpha\beta}.X_1^{\beta\alpha}(y)]\}$$

$$y \in [y_1^*,y_2^*]^{\beta\alpha}$$

where

i) $$U_i(y) = [1 - \frac{Y_2^{\alpha\beta}(X_i^{\beta\alpha}(y)}{X_i^{\beta\alpha}(y)}] [y(\lambda\pi_1 + \frac{\beta}{X_i^{\beta\alpha}(y)}) + (\lambda\pi_2 + \frac{\alpha}{X_i^{\beta\alpha}(y)}) Y_1^{\alpha\beta}(X_i^{\beta\alpha}(y)) - S]$$

$i=1,2$

ii) $V_i(y)$ is simply obtained from U_i by exchanging $Y_2^{\alpha\beta}(.)$ and $Y_1^{\alpha\beta}(.)$

Setting $X_1^{\beta\alpha}(y) = z$ and $\bar{z}$ = "conjugate of z", (4.3) can be rewritten as

$$\boxed{\begin{aligned} &K(z)\ G_1(Y_2^{\alpha\beta}(z)) - K(\bar{z})\ G_1(Y_2^{\alpha\beta}(\bar{z})) \\ &= L(z)\ G_1(Y_1^{\alpha\beta}(z)) - L(\bar{z})\ G_1(Y_1^{\alpha\beta}(\bar{z})) \end{aligned}} \qquad (4.5)$$

$$, \ z \in \phi^{\beta\alpha}$$

where

$$(4.6) \qquad \begin{cases} K(z) = \dfrac{(1 - \dfrac{Y_2^{\alpha\beta}(z)}{z})\Delta(z)}{(1 - \dfrac{Y_1^{\beta\alpha}(z)}{z})\ W^{\alpha\beta}(z)} \\ \\ L(z) \text{ is obtained by exchanging } Y_1^{\alpha\beta}(.) \text{ and } Y_1^{\alpha\beta}(.) \text{ in } K(z). \end{cases}$$

Equation (4.5) describes a Carleman-Hilbert problem (which is of general type, because the contour is not a circle).

Lemma 4.1. Equation (4.5) has a solution iff

$$(4.7) \qquad \frac{d\Delta(x)}{dx}\Big|_{x=1} > 0$$

which is equivalent to

$$(4.8) \qquad \lambda < \alpha + \beta$$

Sketch of the proof. There are three steps :

1) we consider first the "dominant part" of (4.5)

$$(4.9) \qquad K(z).G_1(Y_2^{\alpha\beta}(z)) - K(\bar{z}).G_1(Y_2^{\alpha\beta}(\bar{z})) = 0 , \qquad z \in \phi^{\beta\alpha} .$$

2) We compute the index of (4.9) [6] which - in the present case - must vanish

3) Using the solution of (4.9), we obtain, with the right hand side member of (4.5) an integral equation which, again, has a solution, because it is equivalent to

$$(4.10) \qquad (I - \Theta)G(z) = H(z) , \qquad \text{where } \Theta \text{ is an operator such that } ||\Theta|| < 1.$$

The solution of (4.10) is

$$G(z) = \sum_{i=0}^{\infty} \Theta^i.H(z)$$

[Formulas are omitted here].

The method of solution proposed by the authors applies to general types of functional equations arising in queueing theory. The reader can imagine, nevertheless, the high difficulty of the computations involved when contours are not circles, since conformal mappings into circular domains [according to Riemann's theorem] are needed. (Even for an ellipse the explicit formula is complicated).

REFERENCES

[1] Coffman E.G., and Mitrani I., Selecting a Scheduling Rule that Meets Pre-Specified Response Time Demands, Proceedings, 5 th Symposium on Operating systems Principles, Austin, 1975.

[2] Flatto L. and MacKean H., Two parallel Queues with Equal Servicing Rates, IBM Research Center Watson, Report RC 5916, March 1976.

[3] Fuchs B.A. and Shabat B.V, Functions of a complex Variable and some of their Applications, Volume I, Pergamon Press, 1964.

[4] Cohen J.W., The Single-Server Queue, North Holland Publishing Co., 1969.

[5] Malyshev V.A., An analytical Method in the Theory of Two-Dimensional Positive Random Walks, translated from Sibirskii Mathematicheskii Zhurnal, Vol. 13, n° 6 pp. 1314-1329, November-December 1972.

[6] Muskhelishvili N.I., Singular Integral Equations, P. Noordhoff N.V. - Gronigen Holland Moscow, 1946

[7] Gradshteyn I.S. and Ryzhik I.M., Tables of Integral Series Products, Academic Press, New York and London, 1965.

[8] Mitrani I and Hine J.H., Complete Parameterized Families of Job Scheduling Strategies, Technical Report Series, University of Newcastle upon Tyre, Number 81, October 1975.

[9] Kleinrock L., Queueing systems, II, Wiley, New York, 1976.

[10] Kingman J.F.C., Two Similar queues in Parallel, Ann. Math. Stat., Vol. 32, 1314-1323, 1961.

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

ANALYTIC METHODS FOR MULTIPROCESSOR SYSTEM MODELLING

A. Kurinckx and G. Pujolle
IRIA-LABORIA
78150 Le Chesnay
France

We review some mathematical results on parallel queueing systems and on many servers queues. We develop an approximate general method to solve such systems with global state dependent routing probabilities and service rates. We also build a mathematical model to study the effects of scheduling policies on system performance as mean response times or queue length distributions.

INTRODUCTION

In a recent classification by Fuller and Sieviorek [1], multiprocessors systems have been distributed in three classes : the tightly coupled processors systems (multiprocessors as Illiac IV, Staran,...), the loosely coupled processors systems (computer networks) and intermediate systems. Extensive works have been realised for the first two categories but there are only few studies devoted to intermediate multiprocessors systems.

The aim of this paper is to develop a method to solve models arising in this type of systems. In particular, we are interested in scheduling policies for jobs on the processors.

In the first section we are going to review some mathematical results obtained in the litterature on queueing systems composed of queues in parallel or of one multiserver queue (Figure 1). Moreover we give some extensions of these results and in particular we find conditions for the existence of product form solutions in systems of parallel queues.

In the rest of the paper, we develop a queueing model of a more realistic multiprocessor system. None of the available methods are applicable to the solution of that model; this is because the routing probabilities and the service rates depend on the global system state. For instance the general "join the shortest queue" policy cannot be studied by any exact mathematical method.

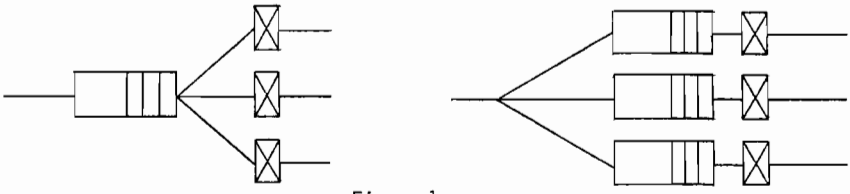
For this reason we introduce an approximate method to solve rather general systems with parallel queues. The method yields approximate values for the marginal probability distributions of queue lengths, the mean numbers of customers, and the mean response times.

The analysis of the model proceeds in two stages : first, using decomposition, the general system is replaced by one consisting of parallel queues where the service rate are function of the number of busy processors; these queues are then analysed by the iterative method to study the effect of scheduling policies on the performance of the system.

Finally we give numerical results for the response times as a function of jobs and system parameters. Several scheduling policies, such as "random access" or "join the shortest queue", are thus investigated and their performance is compared.

MATHEMATICAL MODELS OF MANY-SERVERS QUEUES

We shall decompose the available models in two categories (Figure 1) : those of category A includes just one queue and several parallel servers, those of category B which are composed of several queues in parallel, each with one server.



Category A Figure 1 Category B

In both categories many policies are available to allocate servers to arriving customers. We are going to study the cases where this allocation is dependant on the state of the systems. Other considerations can be added : state dependant service times, possibility to jump from one queue to another one (jockeying). Some studies are devoted to this class of problems, we are going to summerize them, and, in the same time we shall examine some extensions.

We assume that service times and interarrival times are independant and exponentially distributed.

Let λ be the total arrival rate, μ_i be the service rate at station i, i=1,...,M, and let us denote

$$\rho_i = \lambda/\mu_i \text{ and } \rho = \lambda/(\sum_{j=1}^{M} \mu_j)$$

For the sake of simplicity we describe only the two servers cases; extensions to M servers are straighforward.

Models of category A

Case 1 : When a customer arrives then :

a) if both servers are busy then, it waits in a single queue managed by FIFO discipline. The customer at the head of the queue occupies the server that becomes idle first.

b) if only one server is free : it occupies the free channel.

c) if both channels are free : it chooses server 1 with probability Π_1 and server 2 with probability $\Pi_2=1-\Pi_1$.

This case is well known and named "Maître d'Hôtel" if $\Pi_1=\Pi_2=\frac{1}{2}$ and "Maître d'Hôtel with preference" otherwise. The case $\Pi_1=\Pi_2=\frac{1}{2}$ has been studied by Gumbel [3] in the general case of M servers. The general case has been studied by Krishnamoorthi [4] with 2 servers. Moreover Krishnamoorthi has shown that the average waiting time is minimized when $\Pi_i=1$ if i is the faster channel.

Case 2 : Let us assume $\mu_1 \leq \mu_2$, namely the server 1 is slower than server 2. When a customer arrives then :

a) if both servers are busy, it waits in a single queue in order-of-arrival sequence. If server 2 becomes vacant the customer at the head of the queue occupies this server. If server 1 becomes vacant the customer at the head of the queue occupies this server if and only if the total number of customers awaiting service is less than a given value m.

b) if server 2 is free, it occupies it.
if server 1 is free, it joins the queue and the customer at the head of the queue occupies this server if and only if the total number of units awaiting service is less than m.

c) if both channels are free, it chooses server 2. The exact analysis of this system is quite difficult and a solution is proposed by krishnamoorthi [4] when the number of servers is 2. This solution uses a complex algorithm. The value of m which minimized the average waiting time is an open problem. Krishnamoorthi proposes the greater integer less than μ_2/μ_1.

Case 3 : This system is identical to the previous one except in the following point: when the number of customers decreases from m+1 to m, server 1 stops and the customer in service goes to the head of the waiting queue. We provide now a solution (believed to be novel) to this case.

Let n be the number of customers in the waiting room. If $n \leq m$ the service rate is μ_1 and if $n > m$ the service rate is $\mu_1+\mu_2$, (we must have $m \geq 1$).

Let $\rho_1 = \frac{\lambda}{\mu_1}$ and $\rho = \frac{\lambda}{\mu_1+\mu_2}$, we have :

$$p(n) = \begin{cases} \rho_1^m \, p(o) & \text{if } 0 \leq n \leq m \\ \rho_1^m \, \rho^{n-m} \, p(o) & \text{if } n > m \end{cases}$$

In normalizing, we obtain :

$$p(o) = \frac{(1-\rho_1)(1-\rho)}{1-\rho-(\rho_1-\rho)\rho_1^m}$$

The mean number of customers in the system can be written as follows :

$$E = \frac{1-(m+1-m\rho_1)\rho_1^{m+1}}{(1-\rho_1)^2} + \frac{(m+1-m\rho)\rho_1^m}{(1-\rho)^2} \, \rho \quad p(o)$$

The stability condition of this system is obtained writting $p(o) > 0$ [5], namely $\lambda < \mu_1+\mu_2$. This value does not depend on m.

A special case has to be noted : m=1. When there is only one customer in the system, then it occupies server 1; if there is more then one customer, both servers are busy. This system can model a system with two processors which cooperate when only one customer is in the system and which work independently otherwise. The solution of this model can be easily extended to more than two parallel servers.

Models of category B

We recall that a waiting room is associated with each server. In a first step, we shall examine only two queues in parallel. Let λ be the external arrival rate and q_i, i=1,2 the routing probabilities. We denote $\lambda_1=q_1\lambda$, $\lambda_2=q_2\lambda$, $\rho_1=\lambda_1/\mu_1$, $\rho_2=\lambda_2/\mu_2$ and $\rho=\lambda/(\mu_1+\mu_2)$.

Case 0 : The two queues are independant (a Bernoulli switch routes the customers). We have $E_1 = \frac{\rho_1}{1-\rho_1}$, $E_2 = \frac{\rho 2}{1-\rho_2}$ and $E=E_1+E_2$, if E_i is the mean number of customers in queue i.

Case 1 : a) In this system, arriving customers join queue i with a fixed probability q_i ($q_1+q_2=1$)

b) If both channels are free : it chooses server 1 with probability Π_1 and server 2 with probability $\Pi_2=1-\Pi_1$

c) If, at any time, the difference in queue lengths exceeds k, the last customer in the longest queue jokeys to the end of the shortest line. Jokeying is instantaneous.

Such a system has been studied by Koenigsberg [6] when k=1 and is named "Tellers' Windows with Jockeying" (with preference if $\Pi_1 \neq \frac{1}{2}$). Koenigsberg shows that the total mean number in the system is identical to this of case 1 of category A.

Case 2 : In this paragraph we study the system where customers join a queue by a global state dependent switch. Once in the assigned queue, jockeying is forbidden

First, we are going to examine condition for the solution to be in product form for the general case of M parallel processors. Let $\vec{n} = (n_1,n_2,\ldots,n_M)$ be the state vector where n_i is the number of customers in queue and server i. Let $q_i(\vec{n})$ be the routing probability for an incoming customer to enter queue i when the state of the system is $\vec{n}$. Let us note $\vec{n}+1_i=(n_1,n_2,\ldots,n_i+1,\ldots,n_M)$. Local balance equations write as follows :

$$\mu_j \; p(\vec{n}+1_j) = \lambda \; p(\vec{n}) \; q_j(\vec{n})$$

This equation have a solution verifying global balance equation if the following conditions are verified : starting from a state $\vec{n}+1_i+1_j$ to enter state $\vec{n}$, the same value of $p(\vec{n})$ must be found by going either through state $n+1_j$ or through state $n+1_j$. This necessary and sufficient condition can be written : $q_j(\vec{n}+1_i)q_i(\vec{n})=q_i(\vec{n}+1_j)q_j(\vec{n})$, $\forall$ i, $\forall$ j, $\forall$ $\vec{n}$. If routing probabilities verify these condtitions, we obtain a product form solution :

$$p(n_1,\ldots,n_M) = C \prod_{j=1}^{M} \left(\frac{\lambda}{\mu_j}\right)^{n_j} q_j(\vec{n}-1_j)\ldots q_j(\vec{n}-n_j 1_j)$$

If M=2 the condition is

$$\frac{q_1(n_1,n_2)}{q_1(n_1,n_{2+1})} = \frac{q_2(n_1,n_2)}{q_2(n_1+1,n_2)}$$

Some interesting cases can now be treated : if routing probabilities verify : $q_i(\vec{n}) = \frac{n_i}{||\vec{n}||}$ with $||\vec{n}|| = \sum_{i=1}^{N} n_i$ or (see also [7]); $q_i(\vec{n}) = \frac{s_i-n_i}{||\vec{s}-\vec{n}||}$ with $||\vec{s}-\vec{n}|| = \sum_{i=1}^{N} (s_i-n_i)$, etc...

we obtain a product form solution.

Extensions to a service time dependant on the state of its queue and an arrival rate dependent on the total number of customers in the system is possible.

Unfortunately, only the solution of few cases is in product form. For example, the simple scheme consisting in joining the shortest queue has neither a product form nor a closed solution. (A study in truncating the state graph is performed by Chow and Kohler [8] and a diffusion approximation by Foschini and Salz [9]).

Now, if service times are dependant on the state of the entire system, no method is known. Nevertheless, as we shall show it, accurate models of many processors systems need a service time dependant on the total state of the system. It is the reason why we are going to develop a new algorithm yielding a solution of a very

large class of multi servers systems.

Comparisons of the many servers models

Before to conclude this section, we are going to give an idea of the performance obtained by the models described previously. We have chosen as a criterion of performance, the mean number of customers in the system. The results of eight models are shown in table 1. The 7 first results are exact results, the last one is obtained by the algorithm that we shall develop in the next section. It corresponds to the "shortest queue" policy with equal probabilities when queue lenghts are identical. The common model has just 2 servers, the second one with a fixed exponentially service rate $\mu_2=1$, the other with an exponentially service rate varying from 2 to 10 by step 1. The cases of column 6 and 7 correspond to product form solutions where the routing probabilities are defined by :

Case 6 $$q_i(\vec{n}) = \frac{10-n_i}{||10-\vec{n}||} \quad i=1,2$$

Case 7 $$q_i(\vec{n}) = \frac{5-n_i}{||5-\vec{n}||} \quad i=1,2$$

the waiting rooms are limited to 10 and 5 respectively in these two cases.

We can note on these comparisons the important difference between the 8 cases. These models are however too rough to approximate correctly a multi-processors system and it is for this reason that we are going ro develop a new algorithm to take into account more possibilities as global state dependant service times.

mean number in the system

$\lambda=1.5$ ($\lambda_1=1$, $\lambda_2=0.5$) $\mu_2=1$

	1	2	3	4	5	6	7	8
	2 independant queues	Maitre d'Hotel or Tellers'Windows with Jockeying	Maitre d'Hotel with preference Teller's Windows with Jockeying and with preference		Queue 1 preemptive resume m=1	product form solutions		Join the shortest queue
	B.0	A1 B1	A1 $\Pi_1=1$ $\Pi_2=0$	B1 $\Pi_1=0$ $\Pi_2=1$	A3	B2		
$\mu_1 = 2$	2.	1.227	1.191	1.256	0.8	2.391	1.996	1.378
$\mu_1 = 3$	1.43	0.727	0.667	0.769	0.571	2.061	1.663	1.046
$\mu_1 = 4$	1.33	0.563	0.484	0.613	0.379	1.930	1.525	0.890
$\mu_1 = 5$	1.25	0.482	0.389	0.537	0.281	1.860	1.449	0.797
$\mu_1 = 6$	1.2	0.434	0.329	0.492	0.223	1.817	1.403	0.736
$\mu_1 = 7$	1.167	0.402	0.288	0.463	0.184	1.787	1.371	0.692
$\mu_1 = 8$	1.143	0.380	0.257	0.443	0.156	1.766	1.348	0.659
$\mu_1 = 9$	1.125	0.363	0.233	0.428	0.136	1.750	1.330	0.634
$\mu_1 = 10$	1.111	0.350	0.214	0.417	0.12	1.737	1.317	0.614

Table 1

AN APPROXIMATE GENERAL METHOD

In this section we present a general method to solve the queueing systems of category B we have described in the previous sections. The method leads to approximate values of the marginal probability distributions of queue lengths, of mean number of customers and of mean response times for systems which verify the following assumptions :

i) customers arrive into the system according to a Poisson process of parameter λ

ii) the probability that an arriving customer joints queue i ($1 \leq i \leq M$) is $q_i(\vec{n})$, an arbitrary function of the number-in-queue state vector $\vec{n} = (n_1,\ldots,n_i,..,n_M)$. Notice that these functions must verify

$$\sum_{i=1}^{M} q_i(\vec{n}) = 1$$

$\forall \vec{n}$ in the state space

iii) service times of queue i server are independant and identically exponentially distributed with mean $\mu_i^{-1}(\vec{n})$.

The method we propose consists in computing for each queue a fictive arrival rate and a fictive service rate only depending on the number of customers in the queue :

$$\lambda_i(n_i) = \sum_{S} \lambda\, q_i(\vec{n})\, p_i(n_i|\underline{\vec{n}}) \qquad n_i = 0,1,\ldots \tag{1}$$

$$\mu_i(n_i) = \sum_{S} \mu_i(\vec{n})\, p_i(n_i|\underline{\vec{n}}) \tag{2}$$

where

- S is the set of states with a fixed number of customers n_i in queue i.
- $\underline{\vec{n}}$ is a vector of dimension M-1 identical to $\vec{n}$ after removing its ith component.
- $p_i(n_i|\underline{\vec{n}})$ is the conditional probability to have n_i customers in queue i if there are n_j customers in queue j, j=1,M, j≠i.

From the fictive arrival and service rates, it is possible to evaluate the marginal queue length distributions. For queue i :

$$p_i(n_i) = p_i(o) \sum_{i=1}^{n_i} \frac{\lambda_i(i-1)}{\mu_i(i)} \qquad n_i = 1,2,\ldots \tag{3}$$

$p_i(o)$ is given by the normalizing equation

$$\sum_{n_i=0}^{\infty} p_i(n_i) = 1 \tag{4}$$

Notice that to write equation (3), it is necessary to assume that the arrival Poisson flow splits up in M Poisson flows so that each queue behaves like a M/M/1 queue. In the following we assume this property approximately verified although it is true only in particular cases (for example q_i state independant for each i).

The main difficulty is to express the value of $p_i(n_i|\underline{\vec{n}})$. In order to be able to compute $\lambda_i(n_i)$ and $\mu_i(n_i)$ we substitute in (1), (2) $p_i(n_i|\underline{\vec{n}})$ by $p_i(n_i)$, namely in the computation of fictive arrival and service rates we assume the conditional probability to be equal to the marginal one. This simplification is the most important approximation of our method..Although the equality of the two probabilities is true only for systems having a product form solution (see section 2), we shall see later

that our method gives good results in almost all the cases.

With this approximation we have to solve the system composed by equations (1), (2), (3) and (4) by the mean of an iterative scheme. In order to have a finite state space we limit the capacity of each queue i to a value m_i chosen such that the errors due to truncating the distributions are not too important. The algorithm we propose is the following :

begin give an initial value to $p_i(n_i)$ $\begin{cases} i=1,\ldots,M \\ n_i=0,\ldots,m_i \end{cases}$
$r_i(n_i)=0$
while $\bigcup_{i=1}^{M} \sup_{n_i=0,..,m_i} |p_i(n_i)-r_i(n_i)| > \varepsilon$ do

begin $r_i(n_i):=p_i(n_i)$ $i=1,\ldots,M$ $n_i=0,\ldots,m_i$;

for i equal 1 to M do

begin compute $\lambda_i(n_i)$ and $\mu_i(n_i)$, $n_i=0,\ldots,m_i$
from eq. (1) (2);
compute $p_i(n_i)$ $n_i=0,\ldots,m_i$ from eq. (3), (4);

end;

end;

end.

Remarks : - ε is the accepted error on each marginal state probability

- as initial value we adopt

$$p_i(n_i) = \frac{1}{m_i+1}$$

- if $m_i=m$ for each i, storage requirements of this algorithm is proportional to $(m+2)(M+1)$, $\mu_i(\vec{n})$ and $q_i(\vec{n})$ being implemented as internal or external procedures.

The problem of the convergence of the algorithm has not been tackled in a pure mathematical way (one should show that the iterative function is of Lipschitz), but it is easy to see that the iterative function is monotoneous; this property suggests the convergence of our algorithm. Be that as it may, all the experiences we performed show that the solution is obtained after few iterations.

The problem of stability of parallel queues systems with global state dependant service rates is still an open problem. Nevertheless, in some particular cases it is possible to give a sufficient condition of stability. If the stability condition is not satisfied, our algorithm provides the following solution :

$$p_i(n_i) = 0 \quad \text{if} \quad n_i < m_i \qquad i=1,\ldots,M$$

$$p_i(m_i) = 1$$

We are now going to give some details about the accuracy of the method. First notice that exact results are obtained in cases where a product form solution exists.

We performed a great number of experiences in order to compare results obtained by our method with simulation results. These experiences show good agreement : the difference is of the same order of magnitude as the looseness of simulation results.

As an example (Table 1) we give one of the most unfavourable case we met : "join the shortest queue" with equal routing probability when queues are of the same length. We note an accuracy of about 10 % on the mean numbers in the stations

$$\lambda = 1.5 \quad \mu_1(n_1,n_2) = \mu_2(n_1,n_2) = 1.5 \quad \forall\, n_1, \forall\, n_2$$

	p(o)	p(1)	p(2)	p(3)	p(4)	mean number in station 1	mean number in station 2
Algorithm	0.5	.375	.117	.008	.000	.633	.633
Simulation	0.5	.341	.114	.028	.007	.698	.703

Table 1

A SPECIFIC APPLICATION TO MODELLING OF A MULTIPROCESSOR COMPUTER SYSTEM

In this section, we examine the behaviour of a computer system including n distributed processors. The architecture of such a system can be modeled by the idealized queueing network shown in figure 2.

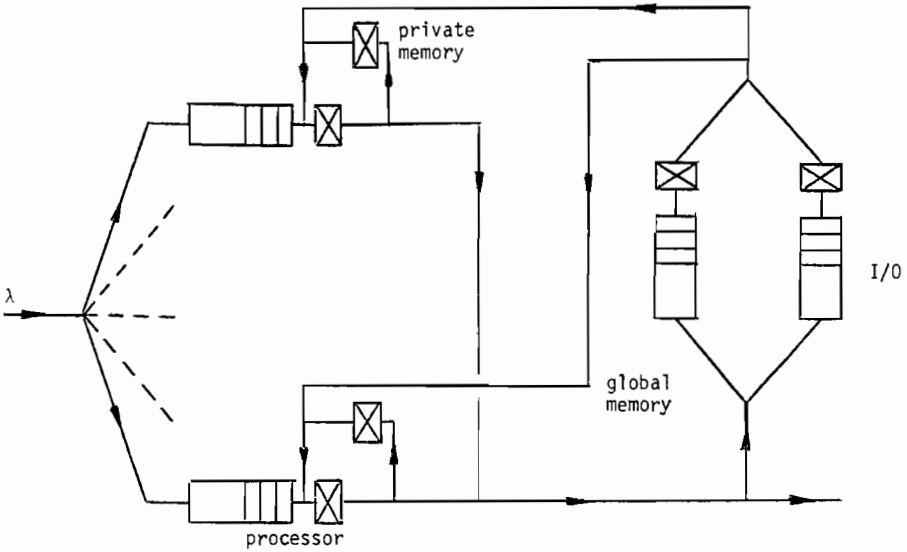


Figure 2

Jobs, which enter the system with rate λ are directed to a processor according to some scheduling policy (for example one of the policies described in the previous sections). The aim of this section is to compare the global mean response times of a job with several policies. In a first step we are going to describe the behaviour of the system.

Jobs are queued behind a processor and are served according to the FIFO discipline

When running a job, a processor can access to informations (instructions and data) contained in its private memory. If the information needed is not present a request is made to a global memory shared among all processors. During its execution, a job can ask for an I/O operation in which case it joins a specialized processor. After completion of a request (I/O or global memory) the execution restarts.

Remarks :
- only one job is loaded in the private memory of a processor
- a job stays in the private memory unitl it has finished its execution
- when a job is queued or served in the I/O or in the global memory device, its processor is blocked.

Firstly, we want to know the time a job spends in a processor before completing, as a function of system parameters (such that : number of processors, size of the memories...) and program behaviour parameters.

Secondly, we want to know the time a job spends in the system from its arrival instant into the queue of a processor to its exit of the system. This time depends on the scheduling policies we use to route the arriving jobs.

To compute the residence time in a processor, we use a decomposition scheme : the idea of the decomposition is to form, with state variables of a complex system, a small number of groups such that :
- interactions of variables, inside a group, can be studied as if external interaction does not exist.
- interactions of groups can be studied without the necessity to take into account interactions between variables inside each group.

In a queueing network, decomposition consists in substituting a subsystem by a simpler entity. The parameters of this entity are computed in the replaced subsystem with a fixed number of customers circulating in it. Systems before and after replacement are equivalent if at steady state :
- probability distributions of the total number of customers in both systems are identical
- probability distributions of state variables which do not belong to the replaced subsystem are identical.

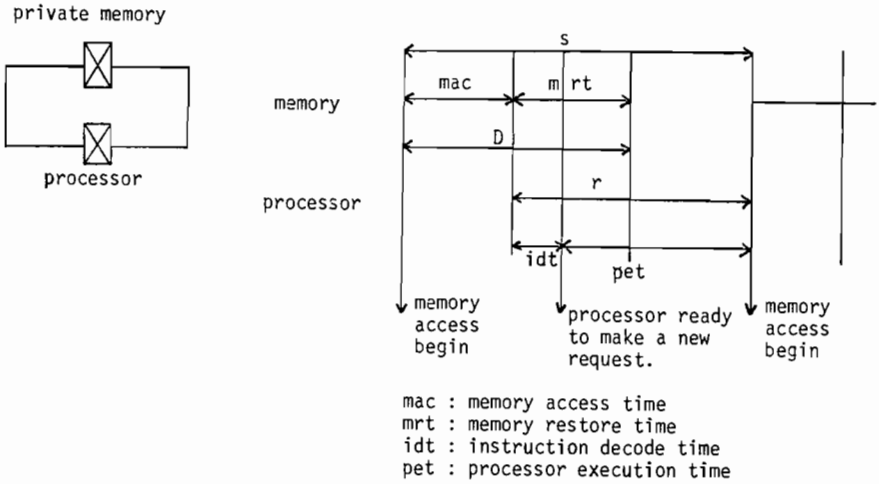
Only few systems have the so-called decomposition property but often it can be shown that the studied system is "near decomposable". The iteractions between the subnetwork to be replaced and other parts of the network. It is the case when internal time constants are at least an order of magnitude smaller than the external one.

Our model shows three levels of decomposability in the terminology of Courtois [2].
- level 1 : user level
- level 2 : processor level
- level 3 : instruction level.

The constants of times of these three levels satisfies for the decomposability conditions.

We are going to study these three levels step by step.

Level 3. We need to know the real busy time of a processor, namely the time during which the processor is either executing, or awaiting for an instruction of its private memory. This level is shown in figure 3 following two points of view (see also [10]).



Assumption : idt+pet ≥ mrt

Figure 3

Let t_e be the total execution time required by a job; the time t_r during which the job occupies the processor at the level 3 is t_e plus the access times plus the decoding times. We assume that one decoding time plus one executing time is greater than one rewriting time. Let s be the mean time between two memory accesses and D be the time of one memory access plus the time to decode the instruction. If we assume D fixed, we have

$$t_r = t_e(\frac{D}{s} + 1)$$

Notice that this time does not include I/O operations and accesses to the global memory.

Level 2. The model of the level 2 is shown in figure 4

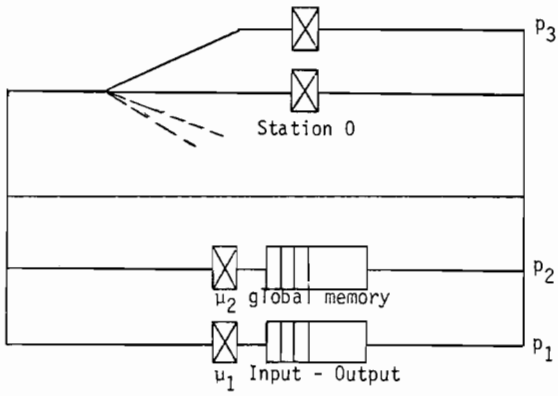


Figure 4

We have to study this closed network with a fixed number (l=1,...,n) of jobs. When a job completes (with probability p_3), it is instantaneously replaced by an other one statistically equivalent. Station 0 is composed by a number of servers equal to number of jobs. By an argument of decomposition the mean response time of a job will be used in level 1 as the mean occupation time of a processor if there are l busy processors. The mean service time of such station 0 has been studied at the level 3. Let a_1 be the number of instructions executed between two I/O requests and a_2 the number of instructions executed between two memory faults (request made to the private memory but insatisfied or direct request to the global memory) and a_3 the total number of instructions of a job.

We assume that a_1, a_2, a_3 are exponentially distributed with parameters ν_1, ν_2, ν_3 respectively, and let a be the number of instructions executed by the processor without interruption (except those of its private memory). We have $a = \min(a_1,a_2,a_3)$. This variable is exponentially distributed with parameter $\nu_o = \nu_1 + \nu_2 + \nu_3$.

Branching probabilities are easily found : $p_1 = \frac{\nu_1}{\nu_o}$, $p_2 = \frac{\nu_2}{\nu_o}$, $p_3 = \frac{\nu_3}{\nu_o}$. Let r be the mean time to execute one instruction, and f be the distribution of the uninterrupted execution time.

From level 3 and from the exponential distribution of a, f is the exponential distribution with mean service time : $\mu_o^{-1} = ar(\frac{D}{s}+1)$.

Let us assume that μ_1 and μ_2 are the rates at which are satisfied I/O requests and global memory requests. The state of the system shown in figure 5 is given by the vector $e = (l_o,l_1,l_2)$ (l_i is the number of customers in station i, $l_o+l_1+l_2=\ell$) and we denote by $p(l_o,l_1,l_2)$ the probability of state e.

From Baskett, Chandy, Muntz, Palacios theorem [11] we have :

$$p(l_o,l_1,l_2) = C \frac{1}{l_o!} \left(\frac{1}{\mu_o}\right)^{l_o} \left(\frac{p_1}{\mu_1}\right)^{l_1} \left(\frac{p_2}{\mu_2}\right)^{l_2}$$

where C is a normalizing constant.

Let W_o, W_1 and W_2 be the mean response times of stations 0, 1, 2 respectively. They can be obtained using BCMP formula. The response time of a job before completion is now :

$$W(\ell) = \frac{W_o}{p_3} + \frac{1}{p_3}\left[\frac{p_2}{p_1+p_2} W_2 + \frac{p_1}{p_1+p_2} W_1\right]$$

This value will be used at level 1 as the occupation time of a processor when there are l busy processors. So before studying level 1, we have to evaluate $W(\ell)$ for l=1,...,n.

Level 1. By the equivalent scheme we substitute the level 2 network by a server of mean service time W(l) that we assume exponentially distributed. The model at level 1 (the user level) is a parallel queueing system of category B, we have studied in section 2. From level 2 we have seen that the service time depends only on ℓ, the number of busy servers when the characteristics of the system are fixed. We find a model which cannot be studied by an exact mathematical technique; so we use the algorithmic approach. Moreover, we shall use scheduling policies, as "join the shortest queue", which require for themselves the algorithmic solution.

We have compared three scheduling policies :

1 - equal routing probability ("random access")

2 - a job entering the system occupies one of the idle processors with equal probability. If all the processors are busy, it joins one of the queues with probability 1/n.

3 - "join the shortest queue" with equal routing probabilities to join one of the shortest queues if more than one.

We assume in our example 5 processors in parallel. If the unit of time is the second we take $\mu_1=\mu_2=100$. The other parameters of the system are $r=10^{-6}$, $s=10^{-5}$, $D=10^{-6}$, $\nu_3=1.5.\ 10^8$ instructions. Now we have treated the example for three values of ν_1 and ν_2.:

1 - $\nu_1 = 10^6$ instructions, $\nu_2 = 5.10^5$ instructions

2 - $\nu_1 = 0.6\ 10^6$ instructions, $\nu_2 = 10^5$ instructions

3 - $\nu_1 = 0.2\ 10^6$ instructions, $\nu_2 = 5.10^4$ instructions

These values correspond to 15s of total execution time and :

1 - 10 I/O requests by second, 20 memory faults by second

2 - 30 I/O , 100

3 - 50 I/O , 200

Figure 5a, 5b, and 5c display the mean response time of a job as a function of the input rate (λ). We also draw in each case a vertical line corresponding to a sufficient (and not necessary) condition of stability :

$$\lambda^* = nW^{-1}(n)$$

When the I/O and global memory requests rates are law (figure 5a), we see that policy number 3 leads always to best performance and response times of second and third policies are very closed. If the input rate is greater than λ^*, the performance of the "join the shortest queue" policy degrades itself more quickly than first policy performance. From this remark one can say that real stability condition of third policy is reached more quickly than with the "random access" policy.

Figure 5b exhibits the same phenomena but brought out more strongly. In particular, beyond λ^* the "join the shortest queue" policy becomes the worst one. When the input rate still increases, the best policy is the first one. Notice that, in this case, the real stability condition of the third policy is very close to sufficient condition λ^*.

Figure 5c leads to the same conclusions. With "bad" jobs (high I/O and global memory request rates), it may be dangerous to use the third policy because response times are very large just beyond λ^*.

The phenomena we have just emphasized are easily explanable. Analysis of level 2 brings out a "thrashing" : performance degrades itself (i.e. the occupation time of a processors increases with "bad" characteristics of jobs (it is due to the congestion of I/O and global memory units). Due to this thrashing it is preferable, in some cases, to let some processors idle. This explains why the "join the shortest queue" policy which shares out very well the load, becomes the worst one with high input rate and "bad" jobs.

To summarize it is surely useful to schedule jobs according to the "join the shortest queue" or the second policy provided that the inputs are controled as soon as the input rate reaches λ^*. An other reason to use this scheduling policy is that the yielded queue length distribution decreases more rapidly than the distribution obtained with the "random access" policy. Figure 6 provides an example of such a

situation. Potential advantage is the reduction of response times dispersion which is highly desirable property in computer systems.

CONCLUSION

After reviewing some mathematical models of parallel queues systems we remarked the bad adequacy of these models to evaluate multiprocessors systems performance. So we developed a rather general algorithmic method to solve parallel queues systems with global state dependant routing probabilities and service rates. This method is sample and yields to accurate results. It can be implemented as a computer program with limited storage requirements.

In the sequel of the paper we described a multiprocessors system with private and global memories and one I/O device. We built a mathematical model and we solved it in two steps. Firstly using the decomposition method we simplified the model to obtain a parallel queueing system. Secondly, we used our algorihtmic method to study the effect on the performance of the scheduling policy for various job behaviours. The main conclusion is the absolutely necessity to implement a dynamic control to take into account the fluctuation of the load. This control will operate on the scheduling policy in order to keep active all the processors when jobs have "good" characteristics and to limit the number of busy processors otherwise.

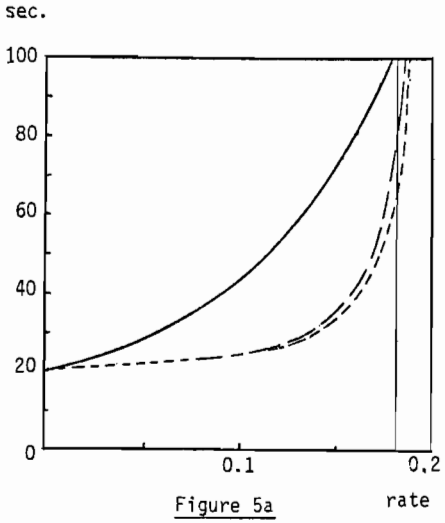


Figure 5a

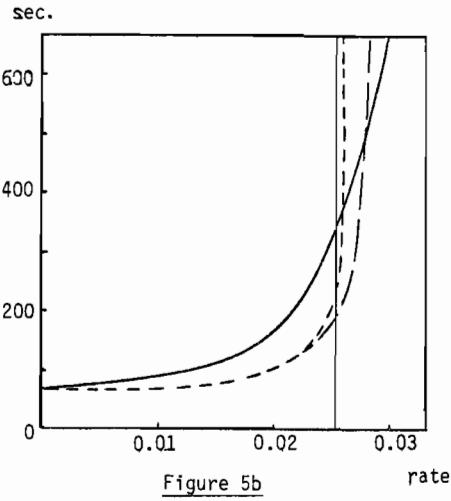


Figure 5b

Mean response time as a function of the input rate λ

policy 1
policy 2
policy 3

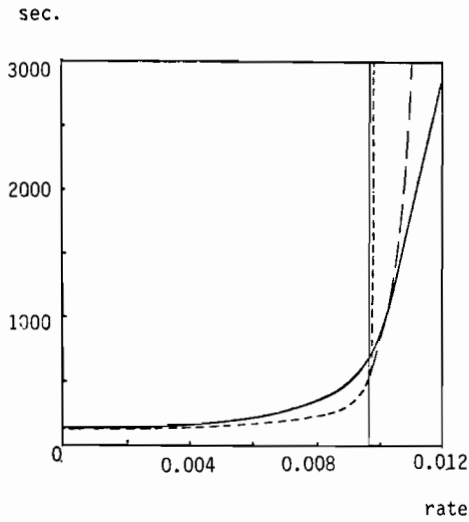


Figure 5c

Mean response time vs. the input rate λ

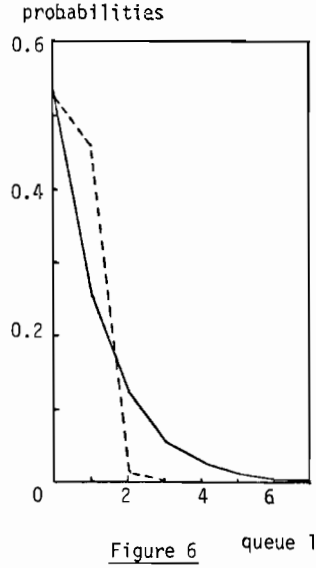


Figure 6

Distribution of the queue length

——— policy 1

- - - - - - - - - - policy 3

REFERENCES

[1] Fuller S.H., Siewioreck D., "Some observations on semi conductor technology and the architecture of large digital modules", Computer 6, 10, 14-21, 1973.

[2] Courtois P.J., "Decomposability - queueing and computer system applications, ACM Monograph series, Academic press, New York, 1977.

[3] Gumbel H., "Waiting lines with heterogeneous servers", Operations Research 8, 504-511, 1960.

[4] Krishnamoorthi B.,"On a Poisson queue with two heterogeneous servers", Operations Research 11, 321-330, 1963.

[5] Pujolle G., "Applications of some Markov chains results to computer systems modelling", IRIA/LABORIA Research Report 289, 1978.

[6] Koenigsberg E., "On jockeying in queues", Management Science, 12, 5, 412-436, 1966.

[7] Glorennec P.Y., Pellaumail J., "Sur une extension de la formule de Jackson, donnant les probabilités stationnaires d'un réseau de files d'attente", IRISA Research Report 23, 1975.

[8] Chow Y.C., Kohler W.H., "Dynamic load balancing in homogeneous two-processor distributed systems", Porc. of the International Symposium on Computer Performance.

[9] Foschini G.J., Salz J., "A basic dynamic routing problem and diffusion", IEEE Trans. on Communications 26, 3, 320-328, 1978.

[10] Bhandarkar D.P., "Analytic models for memory interference in multiprocessor computer systems", Research Report, Carnegie Mellon University, 1973.

[11] Baskett F., Chandy M., Muntz R., Palacios J., "Open, closed and mixed networks of queues with different classes of customers", JACM 22, 2, 1975.

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

THE DISTRIBUTION OF QUEUEING NETWORK STATES AT INPUT AND OUTPUT INSTANTS

K.C. Sevcik* and I. Mitrani**
IRIA Laboria
Rocquencourt, France

Queueing networks are studied at selected points in the steady-state, namely the moments when jobs of a given class arrive into a given node (either from the outside or from other nodes), and the moments when jobs of a given class leave a given node (either for the outside or for other nodes). The processes defined by these points are known to be, in general, non-Poisson, interdependent and serially correlated; therefore the relation between the distribution of the system state embedded at those moments, and the steady-state (or random point) distribution, is not obvious a priori. For a large class of networks having product form equilibrium distributions we show that (a) if the given job class belongs to an open subchain, the state distributions at input points, output points and random points are identical and (b) if the job class belongs to a closed subchain, the distribution at input points is the same as the steady-state distribution of a network with one less job in that subchain.

0. INTRODUCTION

Ever since the discovery (J.R. Jackson, [11]) that in some cases, the steady-state distribution of a network of queues can be factored into a product of the marginal distributions of the states of individual nodes, there has been some uncertainty as to the exact meaning of this result . If a node has c parallel exponential servers, to what extent can it be treated as an isolated M/M/c queue ? In particular, can the distribution of the sojourn times at that node (and not just the mean sojourn time) be obtained by such a treatment ? The answer to this last question depends on whether customers arriving into the node see the steady-state distribution of the node state and that, in turn, has prompted several investigations of the properties of input streams.

---

* on research leave from the University of Toronto

** on leave of absence from the University of Newcastle upon Tyne

Burke [3] has shown, by examining the simplest non-trivial network (a single M/M/1 node with Bernoulli feedbacks), that the input process composed of the external arrivals and the fed-back customers is not Poisson in general. Disney and McNickle [9] have strengthened this result by showing that the input process is not even renewal. On the basis of this and other evidence of the complexity and interdependence of the flow processes in a queueing network, these two authors express a strong doubt about the possibility of decomposing the network and treating individual nodes in isolation.

Yet in the one-node example which Burke considered, it emerged that although the input customers do not form a Poisson stream, they still see the steady-state distribution of the queue size (with the proviso that fed-back customers do not include themselves in the queue that they see). Input instant and steady-state distributions were shown to be closely related also in three finite-state models. These are : an M/M/1 queue with a finite customer population (Cooper [6]), an M/M/1 queue with finite waiting room (Cooper [6]) and a two-node closed network with a feed-back loop around one of the nodes (Mitrani [18]). The first two models can be viewed as two-node closed networks in which the service rate at the second node is, respectively, proportional to the number of customers there or constant. In all three cases the following result holds :

$$P_K^i(n) = \frac{P_K^e(n)}{1-P_K^e(K)} = P_{K-1}^e(n), \quad n=0,1,\ldots,K-1$$

where $P_K^i(n)$ and $P_K^e(n)$ are the input instant and the equilibrium probabilities of n customers at node 1 given that there are K customers in the network.

We shall demonstrate that these results generalise to a large class of networks, open, closed or mixed, with many customer classes, where scheduling disciplines and service requirement distributions are of certain types. The probability that a class r customer joining node m sees network state S, $\pi(S)$, depends on whether the network is open or closed with respect to class r. If open, then $\pi(S)$ is equal to the probability that a class r customer arriving from the outside into node m sees state S (which is equal to the steady-state probability of S if the external arrivals are state-independent). If closed, then $\pi(S)$ is equal to the steady-state probability of S in a network with one less class r customer. Similar results hold for the state left behind by a class r customer leaving node m.

The above probabilities will be derived as the ratio of two rates : the rate at which class r customers join node m and find network state S,

divided by the total rate at which class r customers join node m. In section 1 we give the justification for this approach. Section 2 contains our main result : a theorem establishing a sufficient condition for a network to have input and output instant state distributions related as described to the steady-state distribution. In section 3 we show that a large class of queueing networks, including those defined by Jackson [12], Gordon and Newell [10], Baskett, Chandy, Muntz and Palacios [2], satisfy the sufficient condition. Finally, in section 4 we discuss the possibility of extending the results to a still broader class of networks.

1. GENERAL ARGUMENT.

Let $\{\sigma_n,\tau_n\}$, n=0,1,... be a Markov renewal process : a Markov chain $\{\sigma_0,\sigma_1,..\}$ embedded at moments $t_0 = 0$, t_1, t_2, ..., where $t_{n+1} = t_n+\tau_n$ (n=0,1,...), so that the joint distribution of σ_{n+1} and τ_n depends only on σ_n (and not on $\sigma_0,\ldots,\sigma_{n-1}$, $\tau_0,\ldots,\tau_{n-1}$). The moments t_0, t_1,... will be refered to as "event points".

Suppose that the Markov chain $\{\sigma_n\}$ is irreducible and aperiodic and that the equilibrium probabilities

$$\pi_i = \lim_{n\to\infty} P(\sigma_n=i)$$

exist for all states i=0,1,... . Consider the Markov renewal process on a large time interval of length T; denote by n(T) the total number of event points during that interval, by $n_i(T)$ the number of event points at which $\{\sigma_n\}$ is in state i and by n_{ij} the number of event points between the j-1'st and the j'th point at which $\{\sigma_n\}$ is in state i (j=1,2,...). Since the consecutive occurences of state i (or of any other state) are regeneration points for the Markov renewal process, the numbers n_{i1}, n_{i2}... are independent and identically distributed. Their expected value is therefore equal to

$$E[n_{ij}] = \lim_{T\to\infty} \frac{E[n(T)]}{E[n_i(T)]}. \tag{1}$$

On the other hand, the expected number of points between consecutive occurences of state i (i.e. the expected recurrence time for state i in the Markov chain $\{\sigma_n\}$) is, under our assumptions, equal to $1/\pi_i$. Dividing the numerator and the denominator in the right-hand side of (1) by T and denoting

$$\gamma = \lim_{T\to\infty} \frac{E[n(T)]}{T} \quad ; \quad \gamma_i = \lim_{T\to\infty} \frac{E[n_i(T)]}{T}$$

we can rewrite (1) as

$$\pi_i = \frac{\gamma_i}{\gamma} . \tag{2}$$

In what follows, we shall use (2) to determine the equilibrium distribution $\pi(S)$ of a queueing network state S embedded at the moments when customers of a given class join (or leave) a given node (those will be the event points). We shall write $\pi(S) = \gamma(S)/\gamma$, where γ is the average number of event points per unit time and $\gamma(S)$ is the average number of event points at which the network is in state S, per unit time.

2. GENERAL RESULT.

Consider an arbitrary topology queueing network with M nodes and R customer classes. Customers may change class as they move from node to node. Specifically, a customer of class r, when completing service at node i, goes to node j as a customer of class s with probability $p_{ir,js}$; that job leaves the network with probability $p_{ir,o} = 1 - \sum_{j,s} p_{ir,js}$ (i,j=1,2,...,M; r,s=1,2,...,R). To simplify notation, we shall reclassify the customers, using the pair (i,r) as the new class index. Thus, from now on, "class r" will specify both the current affiliation and the current location of a customer (r=1,2,...,M R). The transition probabilities become p_{rs} (the probability that a class r customer completing service turns into a class s customer) and p_{ro} (the probability that he leaves the network).

The set of customer classes is split by the transition probability matrix into one or more non-intersecting subsets, or "subchains", in the following way : two customer classes belong to the same subchain iff there is a non-zero probability that a customer will enter both classes during his life in the network. Denote these subchains by E_1, E_2,..., E_D (for example, if in the original notation we had $p_{ir,js} = 0$ for $r \neq s$, there would now be at least R subchains).

It may be that some subchains are closed, having a constant number of customers in them at all times, while others are open, with external arrivals and departures. Moreover, the external arrival processes may be state dependent, in a restricted way. Let S be the network state, $N_o(S)$ be the total number of customers in the open subchains and $c_\ell(S)$ be the number of customers in subchain E_ℓ when the network is in state S. The external arrivals may be generated in either, but not both, of the following two ways :

a) By a single non-homogeneous Poisson process whose instantaneous rate, $\lambda(N_o(S))$, depends on the system state via the total number of customers in the open subchains. A new arrival joins class r with probability p_{or}. ($\sum_r p_{or} = 1$)

b) By separate state-dependent Poisson processes, one for each open subchain. The instantaneous rate of the ℓ'th process, $\lambda_\ell(c_\ell(S))$, depends on

S via the number of customers in the subchain E_ℓ. A new arrival in the ℓ'th stream joins class r with probability p_{or} ($\sum_{r\in E_\ell} p_{or} = 1$ if E_ℓ is open).

The flow of customers around the network is described by D systems of linear equations :

$$(3) \qquad e_r = p_{or} + \sum_{s\in E_\ell} e_s p_{sr} \; ; \; r \in E_\ell \; , \quad \ell=1,2,\ldots,D.$$

If $p_{or}\neq 0$ for some $r \in E_\ell$ then the subchain E_ℓ is open; the corresponding system (3) has a unique solution $\{e_r\}$ which can be interpreted as the average number of times a customer joins class r during his life in the network. If $p_{or}=0$ for all $r \in E_\ell$ then E_ℓ is closed; the system (3) has infinitely many solutions (differing from each other by a multiplicative constant) which can be interpreted as the relative frequency with which jobs join class r. The number of customers circulating in a closed subchain is constant.

We shall use the following notation :

$\mathcal{N}$ is the network configuration; it includes the nodes, the customer classes, the subchains and the number of customers in each closed subchain ;

ℓ_r is the index of the subchain to which class r belongs ;

m_r is the index of the node to which class r belongs ;

$\mathcal{N}_{-r}$ is a network configuration which is identical to $\mathcal{N}$ if E_{ℓ_r} is open, and differs from $\mathcal{N}$ by having one less customer in E_{ℓ_r} if E_{ℓ_r} is closed.

S is a network state; it describes the customer configuration at each node : numbers, classes, stages of service reached (a precise definition will be given later);

S_{+r} is the set of all network states which differ from S by having one more customer of class r (the elements of S_{+r} may differ from each other only by the stage of service reached by the extra class r customer and by his position in the queue);

$\nu_r(\sigma,s)$ is the instantaneous rate at which class r customers complete service in state σ and leave behind state S;

$$\Lambda_{\ell_r}(S) = \begin{cases} 1 \text{ if } E_{\ell_r} \text{ is closed} \\ \lambda(N_o(S)) \text{ if } E_{\ell_r} \text{ is open and the external arrivals are of type (a)} \\ \lambda_{\ell_r}(c_{\ell_r}) \text{ if } E_{\ell_r} \text{ is open and the external arrivals are of type (b);} \end{cases}$$

$P_{\mathcal{N}}(S)$ is the steady-state probability of state S in network configuration $\mathcal{N}$; the steady-state distribution is assumed to exist;

$\pi^r_{\mathcal{N}}(S)$ is the equilibrium probability that, in network configuration $\mathcal{N}$, a customer joining class r finds state S. Note that, since the customer does not include himself in the state that he sees, S is not a state of $\mathcal{N}$ but a state of $\mathcal{N}_{-r}$ (the two differ when E_{ℓ_r} is closed);

$\xi^r_{\mathcal{N}}(S)$ is the equilibrium probability that, in network configuration $\mathcal{N}$, a customer completing service in class r, leaves behind state S (similar remark to the above applies).

Our general result can now be stated :

Theorem. If, for each class r in network configuration $\mathcal{N}$,

$$\sum_{\sigma \in S_{+r}} P_{\mathcal{N}}(\sigma)\, \nu_r(\sigma,S) = C\ \Lambda_{\ell_r}(S)\, e_r\, P_{\mathcal{N}_{-r}}(S), \tag{4}$$

where C is a constant independent of S, C=1 if r belongs to an open subchain and $\{e_r\}$, r=1,2,..., MR is any solution of (3), then

$$\pi^r_{\mathcal{N}}(S) = \xi^r_{\mathcal{N}}(S) = \frac{\Lambda_{\ell_r}(S) P_{\mathcal{N}_{-r}}(S)}{\sum_{\sigma \in \mathcal{N}_{-r}} \Lambda_{\ell_r}(\sigma) P_{\mathcal{N}_{-r}}(\sigma)} \tag{5}$$

Proof. The average number of customers who join class r and find state S per unit time, in the steady-state, is equal to

$$\Lambda_{\ell_r}(S)\, p_{or} P_{\mathcal{N}}(S) + \sum_{r' \in E_{\ell_r}} \left\{ \left[\sum_{\sigma \in S_{+r'}} P_{\mathcal{N}}(\sigma) \nu_{r'}(\sigma,S) \right] p_{r'r} \right\}, \tag{6}$$

The first term in (6) counts the customers arriving from outside the network who find state S and join class r (that term is zero if E_{ℓ_r} is closed); the second term includes all service completions in E_{ℓ_r} after which a customer is directed to class r and sees state S. Subsituting (4) into (6) and remembering that if E_{ℓ_r} is open then C=1 and $\mathcal{N}$ coincides with $\mathcal{N}_{-r}$, we rewrite (6) as

$$C\, \Lambda_{\ell_r}(S)\, P_{\mathcal{N}_{-r}}(S) \left[p_{or} + \sum_{r' \in E_{\ell_r}} e_{r'} p_{r'r} \right] = C\, \Lambda_{\ell_r}(S) P_{\mathcal{N}_{-r}}(S) e_r \tag{7}$$

where we have used (3). The total average number of customers who join class r per unit time, in the steady-state, is obtained by summing (7) over all possible states S. Dividing the former average by the latter and cancel-

ing the common factor Ce_r, we obtain equation (5) for $\pi^r_{\mathcal{M}}(S)$. To demonstrate the equation for $\xi^r_{\mathcal{M}}(S)$ we note that the average number of class r customers who complete service and leave behind state S, per unit time, in the steady-state, is equal to

$$\sum_{\sigma \in S_{+r}} P_{\mathcal{M}}(\sigma)\ \nu_r(\sigma,S)$$

which, according to (4), is equal to (7). From this point, the argument is the same as for $\pi^r_{\mathcal{M}}(S)$. □

Let us examine separately the cases when E_{ℓ_r} is open ($\mathcal{M}_{-r} = \mathcal{M}$) and when it is closed ($\Lambda_{\ell_r} = 1$). The theorem states that, in networks which satisfy (4),

(i) All input (output) customers in open subchains see (leave behind) the same distribution of the network state as the customers arriving in those subchains from outside the network. In particular, if the external arrivals into a subchain E_K are state-independent then they all see (leave behind) the steady-state distribution :

(8) $$\pi^r_{\mathcal{M}}(S) = \xi^r_{\mathcal{M}}(S) = P_{\mathcal{M}}(S)$$

for all $r \in E_{\ell_r}$, even though the external arrivals into other subchains may be state-dependent.

(ii) All input (output) customers in closed subchains see (leave behind) the steady-state distribution of a network with one less customer in those subchains :

$$\pi^r_{\mathcal{M}}(S) = \xi^r_{\mathcal{M}}(S) = P_{\mathcal{M}_{-r}}(S)\ ,\quad r \in E_{\ell_r}\quad (E_{\ell_r} \text{ closed}).$$

Moreover, since the configurations $\mathcal{M}_{-r}$ and $\mathcal{M}_{-r'}$ are identical if r and r' belong to the same subchain, the index r in the right-hand side of the last equation may be replaced by any index r' from the same subchain :

(9) $$\pi^r_{\mathcal{M}}(S) = \xi^r_{\mathcal{M}}(S) = P_{\mathcal{M}_{-r'}}(S)\ ,\quad r, r' \in E_{\ell_r}\quad (E_{\ell_r} \text{ closed})$$

Thus, both in open and in closed subchains, the distribution of the system state seen by a customer is independent of which node he is entering (leaving) or what his current class is.

Going back to the original model formulation, with nodes (1,2,...,M) and classes (1,2,...,R), the network state distribution seen by a class r

input into node m depends on whether the pair (m,r) belongs to an open or a closed subchain. Different inputs to the same node may fall under category (i) or (ii), depending on their affiliation. A similar remark applies to outputs.

Note also that, if (5) is true in terms of a detailed network state S, it will obviously be true in terms of various aggregated states; in particular, (i) and (ii) apply to the marginal distributions of individual node states seen (left behind) by input (output) customers. Thus, for example, the distribution of sojourn times at a node in an open Jackson network can be obtained by isolating that node and treating it as an independent M/M/1 or M/M/c queue with the appropriate parameters.

A few words about the meaning of the sufficient condition (4) are in order. If r is in an open subchain, σ and S are states of the same network; in that case (4) appears to equate the instantaneous rate at which state S is entered due to class r service completions, with the instantaneous rate at which S is left due to class r inputs. However, this is only an intuitive interpretation because the input process may not be Poisson. In the extremely special case of a network consisting of one open node without feedback, equations (4) are the local balance equations. In the case of closed subchains the situation is similar, but less intuitive because σ and S are states in different networks.

3. NETWORKS WHICH SATISFY THE SUFFICIENT CONDITION.

In this section we show that equation (4) holds for the large class of queueing networks with product form steady-state distributions defined by Baskett, Chandy, Muntz and Palacios [2].

The nodes of a BCMP network can be of four types, depending on the number of servers and the scheduling strategy employed. These are (1) single server FCFS (first-come-first-served), (2) single server PS (processor-sharing), (3) IS (infinitely many servers, or as many servers as there are customers) and (4) single server LCFSPR (last-come-first-served-preemptive-resume). The required service times at node types 2, 3 and 4 may have arbitrary Coxian distributions (distributions with rational Laplace transforms), with different parameters for the different customer classes; those at type 1 nodes must be exponentially distributed with the same parameter for all classes.

As before, we use a single index to denote the joint (node, class) status of a customer and we call it a class index. If a Coxian distribution is allowed for the required service times of class r customers, its para-

meters are Q_r (number of stages), A_{rj}, $j=1,2,\ldots,Q_r$ (the probability of reaching stage j; $A_{r1}=1$), b_{rj}, $j=1,2,\ldots,Q_r$ (the probability of exiting after stage j; $b_{rQ_r}=1$) and $1/\mu_{rj}$, $j=1,2,\ldots,Q_r$ (the mean of stage j); otherwise the mean of the exponential distribution is $1/\mu_r$ (and it is the same for all r whose node components are the same).

The network state is defined as the vector $S=(S_1,S_2,\ldots,S_{MR})$ whose r'th element describes the state of the class r customers and depends on the type of node m_r : if that type is 2 or 3 then S_r is the vector $(k_{r1},k_{r2},\ldots,k_{rQ_r})$, where k_{rj} is the number of class r customers in the j'th stage of their service ($k_r=k_{r1}+\ldots+k_{rQ_r}$ is the total number of class r customers); if the node type is 1 then S_r is the set of integers $\{h_1,h_2,..,h_{k_r}\}$ indicating the position of each class r customer in the corresponding FCFS queue; if the node type is 4 then S_r is the set of pairs $\{(h_1,j_1),(h_2,j_2),\ldots,(h_{k_r},j_{k_r})\}$ indicating, for each class r customer, his position in the LCFS queue and his stage of service.

Baskett et alia have shown that the stationary distribution (when it exists) of state S in network configuration $\mathcal{N}$ is given by (we have reorganised the expression to conform with present notation)

$$(10)\qquad P_{\mathcal{N}}(S) = \frac{1}{G_{\mathcal{N}}}\, d(S) \prod_{m=1}^{M} q_m(n_m) \prod_{r=1}^{MR} [e_r^{k_r} f_r(S_r)] \quad \forall\ S \text{ feasible in } \mathcal{N}$$

where $G_{\mathcal{N}}$ is a normalisation constant chosen so that (10) is a distribution;

n_m is the number of customers at node m ;

$$d(S) = \begin{cases} 1 & \text{if there are no external arrivals} \\ \prod_{i=0}^{N_a(S)-1} \lambda(i) & \text{if there are external arrival of type (a)} \\ \prod_{\ell=1}^{D} \prod_{i=0}^{c_\ell(S)-1} \lambda_\ell(i) & \text{if there are external arrivals of type (b),} \end{cases}$$

$$q_m(i) = \begin{cases} i! & \text{if node m is of type 2,} \\ 1 & \text{otherwise ;} \end{cases}$$

$$f_r(S_r) = \begin{cases} (1/\mu_r)^{k_r} & \text{if node } m_r \text{ is of type 1,} \\ \prod_{j=1}^{Q_r} [(A_{rj}/\mu_{rj})^{k_{rj}}/k_{rj}!] & \text{if node } m_r \text{ is of type 2 or 3,} \\ \prod_{i=1}^{k_r} (A_{rj_i}/\mu_{rj_i}) & \text{if node } m_r \text{ is of type 4.} \end{cases}$$

The network configuration is reflected in (10) only through the set of feasible states and hence through the normalisation constant $G_{\mathcal{N}}$.

If follows from (10) that if a state σ differs from the state S by the presence of an additional class r customer, then the relation between $P_{\mathcal{N}}(\sigma)$ and $P_{\mathcal{N}_{-r}}(S)$ is, depending on the type of node m_r,

$$P_{\mathcal{N}}(\sigma) = \frac{G_{\mathcal{N}_{-r}}}{G_{\mathcal{N}}} P_{\mathcal{N}_{-r}}(S)\, \Lambda_{\ell_r}(S)\, e_r\, g(r) \tag{11}$$

where

$$g(r) = \begin{cases} 1/\mu_r & \text{if type 1,} \\ [(n_{m_r}+1)/(k_{rj}+1)](A_{rj}/\mu_{rj}) & \text{if type 2,} \\ (A_{rj}/\mu_{rj})/(k_{rj}+1) & \text{if type 3,} \\ (A_{rj}/\mu_{rj}) & \text{if type 4.} \end{cases}$$

and where j is the stage of service reached by the extra customer. The rate of class r service completion in state σ such that state S remains behind, $\nu_r(\sigma,S)$, may depend on the type of node m_r, on the stage of service j reached by the extra class r customer and on his position h in the corresponding queue :

$$\begin{aligned} &\text{type 1,} && \nu_r(\sigma,S) = \begin{cases} \mu_r & \text{if } h=1 \\ 0 & \text{otherwise} \end{cases} \\ &\text{type 2,} && \nu_r(\sigma,S) = \mu_{rj}\, b_{rj}(k_{rj}+1)/(n_{m_r}+1) \\ &\text{type 3,} && \nu_r(\sigma,S) = \mu_{rj}\, b_{rj}(k_{rj}+1) \\ &\text{type 4,} && \nu_r(\sigma,S) = \begin{cases} \mu_{rj} b_{rj} & \text{if } h=1 \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{12}$$

Combining (11) and (12), and bearing in mind that, for all r,

$$\sum_{j=1}^{Q_r} A_{rj}\, b_{rj} = 1 ,$$

we find

$$\sum_{\sigma \in S_{+r}} P_{\mathcal{N}}(\sigma)\, \nu_r(\sigma,S) = \frac{G_{\mathcal{N}_{-r}}}{G_{\mathcal{N}}} \Lambda_{\ell_r}(S)\, e_r\, P_{\mathcal{N}_{-r}}(S).$$

This completes the verification of (4) for these networks, since $G_{\mathcal{N}_{-r}}/G_{\mathcal{N}}$ is independent of S and is equal to 1 if E_{ℓ_r} is open.

Although there is a certain freedom in choosing the values of e_r in closed subchains, the quantity $e_r G_{\mathcal{N}_{-r}}/G_{\mathcal{N}}$ is independent of that choice. That quantity is, in fact, equal to the actual throughput for class r in configuration $\mathcal{N}$ (provided that arrivals into open chains of the same network are state-independent).

The definition of BCMP networks includes also the possibility of state-dependent service rates at either individual nodes or subsets of nodes. The rate of service for class r customers may depend on k_r (except at type 1 nodes) and/or on n_m and/or on Σn_m over a certain subset of nodes (but if there is a dependency on n_m or on Σn_m, it applies to all other classes at that node or nodes). These dependencies are reflected in the product form (10) as factor terms (see [2]). It is a routine task to show that (4) continues to hold.

Thus the theorem of section 2 applies to all BCMP networks.

For closed subchains, the theorem expresses the input and output instant probabilities $\pi^r_{\mathcal{N}}(S)$ and $\xi^r_{\mathcal{N}}(S)$ in terms of the steady-state distribution of network configuration $\mathcal{N}_{-r}$. It may be desirable to have also an expression in terms of the distribution of network $\mathcal{N}$. Such an expression is provided by the sufficient condition (4). Substituting (4) into (9) gives, for closed subchains,

$$(13) \qquad \pi^r_{\mathcal{N}}(S) = \xi^r_{\mathcal{N}}(S) = \left[\sum_{\sigma \in S_{+r'}} P_{\mathcal{N}}(\sigma)\, \nu_{r'}(\sigma, S) \right] / (Ce_{r'})$$

where r' is any customer class in the same subchain as r; $Ce_{r'}$ is the throughput for class r'. If r' happens to have exponentially distributed required service times (with mean $1/\mu_{r'}$) and if the service rate for r' is state-independent, we can simplify (13) :

$$(14) \qquad \pi^r_{\mathcal{N}}(S) = \xi^r_{\mathcal{N}}(S) = P_{\mathcal{N}}(S^*)/\rho_{r'}$$

where S^* is the single element of the set $S_{+r'}$ in which the extra class r' customer is being served, and $\rho_{r'} = Ce_{r'}/\mu_{r'}$ is the utilisation of class r'. The right-hand side of (14) is the conditional probability of the state S^*, given that an r' customer is in service. Equation (14) is a generalisation of the results mentioned in the introduction.

Similar results can be obtained for the state distributions just after input instants or just before output instants. For example, the probability that the network is in state S just after a customer joins class r, for a node with state independent service rate and exponential service times for class r, is equal to $P_{dp}(S)/\rho_r$.

4. EXTENSION TO OTHER NETWORKS.

The class of networks known to have product form has been expanded recently with respect to permissible service disciplines and service requirement distributions. The mathematical (but not practical) limitation that service time distributions must have rational Laplace transforms can be relaxed to allow general absolutely continuous distributions with finite means [1,4,5]. A parameterized family of servicing disciplines can be defined, which includes the four disciplines listed in the last section as special cases [4,5,13].

Our sufficient condition of section 2 is also satisfied by these move general networks. Both Chandy, et al. and Kelly view each queue as having "stations". In general, customers arrive into, receive service in and depart from any station in the queue. A queueing discipline is defined in terms of how an arriving customer selects a station (forcing other customers to move back one position) and what proportion of the server's attention is devoted to each station in each state of the queue. Let $R_m(n_m)$ be the total rate at which service is provided at node m when there are n_m customers there. Then the queue discipline is "exponential local balancing" if there is function $q_m(.)$ such that $R_m(n_m) = \frac{q_m(n_m-1)}{q_m(n_m)}$. Further, an exponential local balancing discipline is "station balancing" if the probability that an arriving customer picks a particular station is equal to the proportion of the server's attention devoted to that station in each possible state of the queue. (The disciplines LCFSPR, PS and IS are station balancing, while FCFS is not).

The system state is defined by the class and remaining service time of the customer at each station of each queue. Whenever all service disciplines are exponential local balancing and, at all nodes, either the discipline is station balancing, or the service requirements for all classes are distributed exponentially with the same mean, the equilibrium state probability density function is given by :

$$(15) \qquad P_{dp}(S) = \frac{\lambda^{N_d(S)}}{G_{dp}} \left[\prod_{m=1}^{M} q_m(n_m) \right] \left[\prod_{r=1}^{R} e_r^{k_r} \prod_{i=1}^{k_r} (1-F_r(W_{ir})) \right]$$

where λ is total (load-independent) arrival rate,

$F_r(.)$ is the distribution function for class r's service requirement,

and W_{ir} is the remaining service time of the i^{th} class r job.

The probability density function of state σ (which has one more class r customer about to complete service) is related to that of state S as follows :

$$P_{\mathcal{A}}(\sigma) = P_{\mathcal{A}_{-r}}(S) \frac{q_m(n_m+1)}{q_m(n_m)} \frac{G_{\mathcal{A}_{-r}}}{G_{\mathcal{A}}} \Lambda_{\ell_r}(S)\, e_r(1-F_r(0^+)) \quad . \tag{16}$$

Since the extra class r customer could be in any position of the queue, and since $P_{\mathcal{A}}(\sigma)$ is the same for all $\sigma \in S_{+r}$, we have

$$\sum_{\sigma \in S_{+r}} P_{\mathcal{A}}(\sigma)\nu_r(\sigma,S) = R_m(n_m+1)\, P_{\mathcal{A}}(S^*) = \frac{q_m(n_m)}{q_m(n_m+1)}\, P_{\mathcal{A}}(S^*)$$

where S^* is any state in S_{+r}.

Observing that $F_r(0^+) = 0$ and using (16) for $P_{\mathcal{A}}(S^*)$ yields :

$$\sum_{\sigma \in S_{+r}} P_{\mathcal{A}}(\sigma)\nu_r(\sigma,S) = \Lambda_{\ell_r}(S) \frac{G_{\mathcal{A}_{-r}}}{G_{\mathcal{A}}} e_r\, P_{\mathcal{A}_{-r}}(S)$$

which is the sufficient condition of section 2.

Thus, if $\pi^r_{\mathcal{A}}(S)$ and $\xi^r_{\mathcal{A}}(S)$ are interpreted as probability density functions rather than probabilities, equations (8) and (9) continue to hold. We make this statement without claiming to have proved it rigorously, since the counting argument of section 1 needs to be generalised in order to cover the case when the quantities involved are infinitesimal.

The product form of the equilibrium state probabilities is retained in certain cases where the routing of customers is not represented as simple transition matrix. Kelly [14] considers the case in which each customer has a specific route which he follows through the network. Kobayashi and Reiser [15] allow the transitions among service centers to be governed by a Markov process of arbitrary (finite) order. The technique used in both cases is to introduce additional "artificial" classes (typically a lot of them) in order to retain more information about the past or future path of the customer. Then, in terms of the enlarged set of classes, all transitions among service centers and non-artificial classes can be represented by a simple transition matrix (or first order Markov process). The theorems of section 2 apply to the detailed states involving the artificial classes, so we only

have to aggregate states by eliminating the distinctions among the artificial classes that correspond to an actual class in order to see that the theorems of section 2 apply also to models with these more general routing patterns.

The product form of the equilibrium probability distribution is also retained when constraints on the customer populations in various subchains is more general than having each chain contain either a constant (closed) or an unlimited (open) number of customers. Lam [16] studies "loss" and "trigger" functions for each customer class based on the number of customers currently in each chain. His work is a generalization of the ideas used by Jackson to unify open and closed networks [12], and by Reiser and Kobayashi to create "semi-closed" subchains, in which the number of customers is allowed to vary between minimum and maximum values, but independently of the populations in other subchains [19].

Lam proves a sufficient condition for the equilibrium state probabilities to retain the same product form as for the networks described by Baskett et al. [2]. Let $T_{\ell}(\sigma)$ be the "trigger" function whose value is zero if the departure of a chain ℓ customer in state σ triggers the arrival of a new chain ℓ customer, and is one otherwise. Similarly, let $L_{\ell}(S)$ be the "loss" function whose value is zero if chain ℓ arrivals in state S are lost and is one otherwise. Then Lam's sufficient condition is

$$T_{\ell_r}(\sigma) = L_{\ell_r}(S) \quad \text{for all } r \text{ and } \sigma \in S_{+r}$$

Thus, an arrival is triggered to stop a change in subchain population if and only if arrivals that would cause the reverse change are lost. Because systems with loss and trigger functions satisfying the above equation have the same product form solution discussed in section 4, they satisfy the sufficient condition of the theorem in section 2. However, the expression (6) indicating the rate with which class r customers find state S must be modified to include the trigger functions :

$$(17)\ \Lambda_{\ell_r}(S) L_{\ell_r}(S) p_{or} P_{\mathcal{N}_{-r}}(S) + \sum_{r' \in E_{\ell_r}} \left\{ \left[\sum_{\sigma \in S_{+r'}} P_{\mathcal{N}}(\sigma) \nu_{r'}(\sigma, S) \right] \left[p_{r'r} + p_{r'o}(1 - T_{\ell_{r'}}(\sigma)) p_{or} \right] \right\}$$

When $L_{\ell_r}(S)=1$, $T_{\ell_{r'}}(\sigma)=1$ because r and r' are in the same subchain and the expression reduces directly to that examined in the proof in section 2. When $L_{\ell_r}(S) = T_{\ell_r}(\sigma)=0$ the expression above again reduces to

$$\Lambda_{\ell_r}(S) P_{\mathcal{N}_{-r}}(S) \frac{G_{\mathcal{N}_{-r}}}{G_{\mathcal{N}}} e_r$$

after some algebraic manipulation and use of the identity

$$\sum_{r_1 \epsilon E_j} (1 - \sum_{r_2 \epsilon E_j} p_{r_1 r_2}) e_{r_1} = 1$$

Thus if the loss and trigger functions satisfy Lam's sufficient condition for a product form solution to exist, then the theorem of section2 remains applicable (although its proof must be generalised as above).

5. CONCLUSIONS.

We have considered queueing networks with multiple customer classes, Markov routing chains of arbitrary order, state dependent arrival and service rates, general service time distributions at service centers with station balancing disciplines, and general constraints on the populations of the routing chains. The presence of even a single feedback loop is such networks causes the input processes at nodes to be non-Poisson and, in general, non-renewal. Yet we have found that these networks with product form equilibrium state distributions have input and output instant distributions related simply to the equilibrium distribution of the same network but with the population constraints (if any) on one chain decreased by one customer. Thus, with negligible additional computation, existing algorithms for calculating the equilibrium state probabilities can also calculate the distribution at instants just before inputs or arrivals or just after outputs or departures.

Among networks with equilibrium distributions that do not have product form, the simple relationships between input instant and equilibrium distributions seem to be retained only in very special cases. The M/Er-2/1 FCFS queue with feedback is perhaps the simplest network without product form equilibrium probabilities. In this case, the input instant and the steady-state distributions are not the same because fed-back customers never see the second service stage busy, while the equilibrium utilisations of the two stages are equál. However, if we consider aggregate states, n, defined as the number of customers at the service center, then for an FCFS M/G/1 queue with feedback, the distributions of n at input instants and in equilibrium are the same. This can be shown by modifying the approach in [9] so that fed-back customers do not see themselves.

However, in a two node closed network where the service requirement distribution is highly skewed, the distribution of n at input instants is not the same as the equilibrium distribution of n in the same closed network

with one less customer. We conjecture that (8) and (9) fail to hold in any network where a FCFS non-exponential node contributes to the input of another node.

REFERENCES

I. Barbour, A.D., Networks of Queues and the Method of Stages. Advanced Applied Probability 8, 3 (September 1976), 584-91.

2. Baskett, F., Chandy, K.M., Muntz, R.R., and Palacios, F.G., Open, closed and mixed networks of queues with different classes of customers. JACM 22, 2 (April 1975), 248-60.

3. Burke, P.J., Proof of a conjecture on the interarrival-time distribution in an M/M/1 queue with feedback. IEEE-TC 24, 5 (May, 1976), 575-76.

4. Chandy, K.M., Howard, J.H., and Towsley, D.F., Product form and local balance in queueing networks. JACM 24, 2 (April 1977), 250-63.

5. Chandy, K.M., Recent work in preparation.

6. Cooper, R.B., Introduction to queueing theory . MacMillan Publ. Co., New York (1972).

7. Cox, D.R., A use of complex probabilities in the theory of stochastic processes. Proc. Cambridge Phil. Soc. 51 (1955), 313-19.

8. Disney, R.L., Random flow in queueing networks : a review and critique. AIIE Transactions 7, 3 (September 1975) 268-88.

9. Disney, R.L. and McNickle, D.C., The M/G/1 queue with instantaneous Bernoulli feedback. Rep. 77-4, Dept. of Ind. and Oper. Eng., University of Michigan, Ann Arbor (April 1977).

10. Gordon, W.J., and Newell, G.F., Closed queueing systems with exponential servers. Op. Res., 15 (1967), 254-65.

11. Jackson, J.R., Networks of waiting lines. Op. Res. 5, (1957), 518-21.

12. Jackson, J.R., Jobshop-like queueing systems. Mgmt. Sci. 10, 1 (October 1963), 131-42.

13. Kelly, F.P., Networks of queues with customers of different classes. J. Appl. Prob. 12 (1975), 542-54.

14. Kelly, F.P., Networks of queues, Adv. Appl. Prob. 8, 2 (June 1976), 416-32.

15. Kobayashi, H. and Reiser, M., On generalization of job routing behavior in a queueing network model.
IBM research report RC5252, Yorktown Heights, (February 1975).

16. Lam, S.S., Queueing networks with population size constraints.
IBM J. R & D. 21, 4 (July 1977), 370-78.

17. Lemoine, A.J., Networks of queues - a survey of equilibrium results.
Mgmt. Sci. 24, 4 (December 1977), 464-81.

18. Mitrani, I., Non priority multiprogramming systems under heavy demand conditions - cutomers viewpoint
JACM 19, 3 (July 1972), 445-452.

19. Reiser, M. and Kobayashi, H., Numerical solution of semi-closed exponential server queueing networks.
Proc. 7th Asilomar Conf. (1973), 308-12.

APPROXIMATE METHODS

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

MULTICLASS OPERATIONAL ANALYSIS OF QUEUEING NETWORKS

J.D. Roode*
Department of Computer Science
Rand Afrikaans University
Johannesburg, South Africa

A multiclass queueing network model is presented as an extension of the work of Denning and Buzen on queueing networks. The model is developed using operational analysis and only the three basic operational principles are invoked. No stochastic assumptions are made and only readily available operational data are used as input to the model. The network balance equations are derived and the product form of solution presented. Finally, algorithms are developed to calculate all performance quantities of interest.

1. INTRODUCTION

In a recent paper, Denning and Buzen [DENN77] showed how operational analysis, developed by Buzen [BUZE76], can be extended to apply to queueing networks - specifically in the context of the study of the performance of multiple-resource computer systems. Using only a few basic assumptions they showed that the usual balance equations for the state of the network can be derived, permitting the well-known product form of solution. When applied to a specific network, their analysis leads essentially to the Gordon-Newell [GORD67] results. In this paper we extend the work of Denning and Buzen to take into account different classes of customers in networks. The model is compared with stochastic multiclass models [BASK75] in section 5.

One of the main benefits arising from the use of operational analysis is that only precisely measurable quantities are used while no assumptions are made which the analyst cannot validate. As a result, the analyst can concentrate his thoughts on the operational level. Thus the concept of distinct customer classes in the operational analysis of queueing networks should be tailored to the actual use the analyst would make of such classes.

Consider a typical computer system supporting batch and timesharing tasks, running under a virtual storage operating system. In such an environment the natural approach for an analyst using a multiclass model would be to consider the different classes of jobs or tasks which would run on the system, e.g. batch jobs, input spooling requests, output spooling requests and timesharing requests. Associated with each job class is a number of system tasks; if for example, the class of batch jobs is considered, the following associated system tasks can be distinguished: paging, pre-emption, channel program building, completion interrupt handling, etc.

---

*This work was done while the author was on sabbatical leave at the National Research Institute for Mathematical Sciences of the CSIR, P O Box 395, Pretoria 0001, South Africa.

Instead of regarding each different job class and its associated system tasks as an ergodic subchain of a two-dimensional Markov chain, a more natural approach is to introduce the concept of class groups and develop the model in such a way that the analyst is required to specify the different class groups - each consisting of a main job class and its related or associated system job classes. As will become clear in the subsequent treatment of the model, the concept of class changes and the associated transition probabilities enter into the development of the model only on the conceptual level. These probabilities need not therefore be specified - instead, only the request throughput rate (or visit ratios) of each centre for each customer class is needed.

The presentation in the following sections is organized as follows. In Section 2 we present a generalization of the work by Denning and Buzen to derive the local balance equations for multiclass queueing networks. The discussion is limited to networks with only one class group - a restriction which is easily relaxed. In Section 3 we address the problem of calculating the normalizing constant G, and derive an algorithm to calculate G recursively. Various descriptive measures of the network are discussed in Section 4 and computational aspects treated - in particular, algorithms are derived for calculating the utilization of each centre and the average queue length there. Multiclass operational network models are compared with their stochastic counterparts in Section 5.

2. OPERATIONAL QUANTITIES AND BALANCE EQUATIONS IN MULTICLASS NETWORKS

In this section we follow to a large extent the notation used by Denning and Buzen [DENN77]. Although the material presented is a fairly straightforward generalization of their treatment of queueing networks, we present full details in order to keep this presentation self-contained.

Consider a computer system consisting of M devices (processors, service centres). Jobs in the system may belong to any one of a finite number of classes. The collection of classes constitutes a class group which consists of a main job class and a number of associated system job classes. Let the classes be numbered 1,2,...,R.

Let n_{ir} be the number of jobs (customers) of class r present at centre i. Then

$n_i = \sum_{r=1}^{R} n_{ir}$ is the total number of customers present at centre i, and

$W_r = \sum_{i=1}^{M} n_{ir}$ is the total number of class r customers in the system.

$N = \sum_{i=1}^{M} n_i$ is the total number of customers in the system. If N is fixed, the system is closed, and if $0 \leq N \leq \infty$ holds, the system is open. During an observation period [0,T], suppose the following data are collected:

$A_{ir}(n)$: number of arrivals of class r customers at centre i, when $n_{ir} = n$

$C_{ir}^{js}(n)$: number of times a customer of class r requests service at centre j as a class s customer immediately after completing a service request at centre i, when $n_{ir} = n$

$T_{ir}(n)$: total time during which $n_{ir} = n$

B_{ir} : total busy time of device i for class r customers.

If we treat the outside world as centre o, then

$C_{or}^{ir}(n)$: number of customers of class r whose first service request is for centre i when $W_r = n$ (no class changes occur on entry to the system)

$C_{ir}^{or}(n)$: number of class r customers whose last service request is for centre i, when $n_{ir} = n$ (no class changes occur on exit from the system)

The number of completions of class r jobs at centre i when $n_{ir} = n$, is computed as

$$C_{ir}(n) = \sum_{j=o}^{M} \sum_{s=1}^{R} C_{ir}^{js}(n).$$

The number of arrivals of class r customers at the system when $W_r = n$, is

$$A_{or}(n) = \sum_{i=1}^{M} C_{or}^{ir}(n).$$

2.1 OPERATIONAL QUANTITIES

Define the following operational quantities:

$X_{ir}(n)$, request completion rate for class r customers at centre i when

$$n_{ir} = n : X_{ir}(n) = \frac{C_{ir}(n)}{T_{ir}(n)}$$

$P_{ir}(n)$, proportion of time when $n_{ir} = n$: $P_{ir}(n) = \frac{T_{ir}(n)}{T}$

$S_{ir}(n)$, the service function for class r customers at centre i when

$$n_{ir} = n : S_{ir}(n) = \frac{T_{ir(n)}}{C_{ir(n)}} .$$

It should be noted that the service function for class r customers may include time devoted by the i-th device to other classes of customers.

The total number of completions for class r customers at centre i is

$$C_{ir} = \sum_{n>o} C_{ir}(n)$$

and the overall request completion rate for class r customers at centre i is $X_{ir} = \frac{C_{ir}}{T}$. It readily follows from the definitions that

$$X_{ir} = \sum_{n>0} P_{ir}(n)\, X_{ir}(n).$$

The mean service time over all class r completions at device i is now given by

$S_{ir} = \dfrac{B_{ir}}{C_{ir}}$ and the utilization of centre i by class r customers is

$U_{ir} = \dfrac{B_{ir}}{T} = X_{ir}\, S_{ir}$. Independent of class, the utilization of centre i is $\sum_{r=1}^{R} U_{ir}$.

Let J_{ir} denote the total job-seconds accumulated at centre i by class r customers, i.e.

$J_{ir} = \sum_{n>0} nT_{ir}(n)$, and let $\bar{n}_{ir}$ be the average number of customers of class r at centre i.

Then $\bar{n}_{ir} = \sum_{n>0} P_{ir}(n)\cdot n = \dfrac{J_{ir}}{T}$. If R_{ir} denotes the average response time to a request by class r customers at centre i, we have $R_{ir} = \dfrac{J_{ir}}{C_{ir}}$.

Hence $\bar{n}_{ir} = R_{ir} \cdot X_{ir}$. (Operational Little's Formula)

Now consider the routing of customers through the network.

Define $q_{ir}^{js} = \dfrac{1}{C_{ir}} \sum_{n>0} C_{ir}^{js}(n)$

where q_{ir}^{js} is the fraction of completions of class r jobs at centre i which are followed immediately by requests, as class s customers, for service at centre j.

2.2 PRINCIPLE OF JOB FLOW BALANCE

The Principle of Job Flow Balance [DENN78] implies the following. For each centre i, X_{ir} is the same as the total input rate of class r customers to centre i. Therefore, if job flow is balanced, we refer to X_{ir} as centre <u>throughputs</u>.

Therefore $C_{js} = A_{js} = \sum_{i=0}^{M} \sum_{r=1}^{R} C_{ir}^{js}$, $C_{ir}^{js} = \sum_{n>0} C_{ir}^{js}(n)$.

Since $q_{ir}^{js} \cdot C_{ir} = C_{ir}^{js}$

it follows that $C_{js} = \sum_{i=0}^{M} \sum_{r=1}^{R} C_{ir}\, q_{ir}^{js}$.

Dividing by T, we obtain the following.

2.3 JOB FLOW BALANCE EQUATIONS

$$X_{js} = \sum_{i=o}^{M} \sum_{r=1}^{R} X_{ir}\, q_{ir}^{js} . \qquad \begin{matrix} j = 0,1,\ldots,M \\ s = 1,2,\ldots,R . \end{matrix}$$

If the network is open, X_{os} for each s will have a value determined by the exogenous environment, and these equations can then be solved for X_{ir} . For a closed system, X_{os} is unknown and the equations have no unique solution : Consider

$$\sum_{s=1}^{R} \sum_{j=1}^{M} X_{js} = \sum_{i=o}^{M} \sum_{r=1}^{R} X_{ir} \left(\sum_{j=1}^{M} \sum_{s=1}^{R} q_{ir}^{js} \right)$$

$$= \sum_{i=o}^{M} \sum_{r=1}^{R} \frac{X_{ir}}{C_{ir}} \left(\sum_{j=1}^{M} \sum_{s=1}^{R} C_{ir}^{js} \right)$$

$$= \sum_{i=o}^{M} \sum_{r=1}^{R} \frac{X_{ir}}{C_{ir}} \left(C_{ir} - \sum_{s=1}^{R} C_{ir}^{os} \right)$$

$$= \sum_{i=o}^{M} \sum_{r=1}^{R} X_{ir} - \sum_{i=o}^{M} \sum_{r=1}^{R} \sum_{s=1}^{R} X_{ir}\, q_{ir}^{os} ,$$

so that $\sum_{s=1}^{R} X_{os} = \sum_{s=1}^{R} \sum_{i=o}^{M} \sum_{r=1}^{R} X_{ir}\, q_{ir}^{os} = \sum_{i=o}^{M} \sum_{r=1}^{R} X_{ir} q_{ir}^{or} .$

Now define $V_{ir} = \dfrac{V_{ir}}{X_{or}}$, which is the flow of customers in class r through centre i relative to the system throughput for class r customers. Then

$V_{ir} = \dfrac{C_{ir}}{C_{or}}$, and V_{ir} is the mean number of completions in class r at centre i for each completion in class r at the system. This means that V_{ir} is the average number of service requests (or visits) per customer in class r at centre i. Following Denning and Buzen, V_{ir} is called the visit ratio of class r customers at centre i. The visit ratio equations now follow directly from the job flow balance equations:

$$V_{or} = 1 \qquad r = 1,\ldots,R$$

$$V_{js} = \sum_{r=1}^{R} q_{or}^{js} + \sum_{i=1}^{M} \sum_{r=1}^{R} V_{ir}\, q_{ir}^{js} . \qquad \begin{matrix} j = 0,1,\ldots,M \\ s = 1,\ldots,R \end{matrix}$$

2.4 STATE OCCUPANCIES AND STATE TRANSITION BALANCE

The state of the system is described by the vector

$$\underline{n} = (\underline{n}_1, \ldots\ldots, \underline{n}_M)$$

where $\underline{n}_m = (n_{m1}, n_{m2}, \ldots\ldots, n_{mR})$

Let $T(\underline{n})$ be the total time during which the network is in state $\underline{n}$ during the interval $[0,T]$. The time proportion for $\underline{n}$ is

$$p(\underline{n}) = \frac{T(\underline{n})}{T}, \quad \text{and} \quad \sum_{\underline{n}} p(\underline{n}) = 1,$$

where the summation is over all possible $\underline{n}$. Let $\underline{k}$, $\underline{n}$ and $\underline{m}$ denote distinct system states, and let $Q(\underline{n},\underline{m})$ be the number of one-step (i.e. without passing through any intermediate state) transitions observed from state $\underline{n}$ to state $\underline{m}$, with $Q(\underline{n},\underline{n}) = 0$. The Principle of Job Flow Balance now implies the Principle of State Transition Balance:

The number of entries to every state is the same as the number of exits from that state during the observation period:

$$\sum_{\underline{k}} Q(\underline{k},\underline{n}) = \sum_{\underline{m}} Q(\underline{n},\underline{m}) \qquad \text{all } \underline{n}.$$

Define the transition rate from $\underline{n}$ to $\underline{m}$ as follows:

$$H(\underline{n},\underline{m}) = \frac{Q(\underline{n},\underline{m})}{T(\underline{n})} \qquad T(\underline{n}) \neq 0.$$

Then the state transition balance equations can be written as

$$\sum_{\underline{k}} T(\underline{k})\, H(\underline{k},\underline{n}) = T(\underline{n}) \sum_{\underline{m}} H(\underline{n},\underline{m}),$$

so that

$$\sum_{\underline{k}} p(\underline{k})\, H(\underline{k},\underline{n}) = p(\underline{n}) \sum_{\underline{m}} H(\underline{n},\underline{m})$$

for all $\underline{n}$ for which each $H(\underline{n},\cdot)$ is defined. Adding the normalizing condition $\sum_{\underline{n}} p(\underline{n}) = 1$ and noting that $p(\underline{n}) = 0$ for those $\underline{n}$ not included in the above balance equations, a unique set of $p(\underline{n})$ will satisfy the equations.

Before attempting to solve the balance equations, we introduce the following simplifying assumption about so-called 'one step behaviour': The only observable state changes result from single customers either entering the system, or exiting from the system, or moving between pairs of centres in the system with accompanying class changes. This assumption implies that the 'neighbour' states of state $\underline{n}$ are

$$\underline{n}_{ir}^{js} = (n_{11},\ldots\ldots,n_{ir}+1,\ldots\ldots,n_{js}-1,\ldots\ldots n_{MR})$$

$$\underline{n}_{ir}^{or} = (n_{11},\ldots\ldots,n_{ir}+1,\ldots\ldots,m_{MR})$$

$$n_{or}^{jr} = (n_{11},\ldots\ldots,n_{jr}-1,\ldots\ldots,n_{MR}).$$

Then for all $\underline{n}$

$$\sum_{\substack{i,j\\r,s}} p(\underline{n}_{ir}^{js})\, H(\underline{n}_{ir}^{js},\underline{n}) + \sum_{i,r} p(\underline{n}_{ir}^{or})\, H(\underline{n}_{ir}^{or},\underline{n}) + \sum_{j,r} p(\underline{n}_{or}^{jr})\, H(\underline{n}_{or}^{jr},\underline{n})$$

$$= p(\underline{n}) \left\{ \sum_{\substack{i,j\\r,s}} H(\underline{n},\underline{n}_{js}^{ir}) + \sum_{i,r} H(\underline{n},\underline{n}_{or}^{ir}) + \sum_{j,r} H(\underline{n},\underline{n}_{jr}^{or}) \right\} .$$

The first terms on the left and on the right correspond to customers making (i,r), (j,s) transitions; the second terms on the left and on the right correspond to customers exiting the system from centre i; the third terms on the left and on the right correspond to jobs entering the system at centre j. Sums over i and j extend over 1,...,M; sums over r and s extend over 1,...,R. For a closed system, the second and third terms on the left and on the right are dropped, and q_{ir}^{js} is increased by $q_{ir}^{or} \cdot q_{os}^{js}$.

In order to solve the balance equations we now have to express the state transition rates in terms of measurable parameters. Consider, for example, the rate $H(\underline{n}_{ir}^{js}, \underline{n})$. This is approximated as follows:

$$H(\underline{n}_{ir}^{js}, \underline{n}) = \frac{Q(\underline{n}_{ir}^{js}, \underline{n})}{T(\underline{n}_{ir}^{js})} \approx \frac{C_{ir}^{js}(n_{ir}+1)}{T_{ir}(n_{ir}+1)} \approx \frac{q_{ir}^{js} C_{ir}(n_{ir}+1)}{T_{ir}(n_{ir}+1)} = \frac{q_{ir}^{js}}{S_{ir}(n_{ir}+1)} .$$

Two major assumptions are made in this approximation. We shall comment on that in Section 5. Following Denning and Buzen, we call these approximated transition rates homogeneous rates and obtain the following.

Customer Transition	State Transition	Homogeneous Rate
(i,r) → (j,s)	$\underline{n}_{ir}^{js} \to \underline{n}$	$H(\underline{n}_{ir}^{js},\underline{n}) = q_{ir}^{js}\, I_{js}/S_{ir}(n_{ir}+1)$
	$\underline{n} \to n_{js}^{ir}$	$H(\underline{n},\underline{n}_{js}^{ir}) = q_{ir}^{js}\, I_{ir}/S_{ir}(n_{ir})$
(i,r) → o	$n_{ir}^{or} \to \underline{n}$	$H(\underline{n}_{ir}^{or},\underline{n}) = q_{ir}^{or}/S_{ir}(n_{ir}+1)$
	$\underline{n} \to \underline{n}_{or}^{ir}$	$H(\underline{n},\underline{n}_{or}^{ir}) = q_{ir}^{or}\, I_{ir}/S_{ir}(n_{ir})$

Customer Transition	State Transition	Homogeneous Rate
$o \to (j,r)$	$\underline{n}_{or}^{jr} \to \underline{n}$	$H(\underline{n}_{or}^{jr},\underline{n}) = X_{or}\, q_{or}^{jr}\, I_{jr}$
	$\underline{n} \to \underline{n}_{jr}^{or}$	$H(\underline{n},\underline{n}_{jr}^{or}) = X_{or}\, q_{or}^{jr}$

where $I_{ir} = \begin{cases} 1 \text{ if } n_{ir} > 0 \\ 0 \text{ if } n_{ir} = 0. \end{cases}$

The homogenized balance equations are now

$$\sum_{\substack{i,j \\ r,s}} p(\underline{n}_{ir}^{js}) \cdot \frac{q_{ir}^{js}\, I_{js}}{S_{ir}(n_{ir}+1)} + \sum_{i,r} p(\underline{n}_{ir}^{or}) \cdot \frac{q_{ir}^{or}}{S_{ir}(n_{ir}+1)} + \sum_{j,r} p(\underline{n}_{or}^{jr})\, X_{or} q_{or}^{jr} I_{jr}$$

$$= p(\underline{n}) \left\{ \sum_{\substack{i,j \\ r,s}} \frac{q_{ir}^{js}\, I_{ir}}{S_{ir}(n_{ir})} + \sum_{i,r} \frac{q_{ir}^{or}\, I_{ir}}{S_{ir}(n_{ir})} + \sum_{j,r} X_{or}\, q_{or}^{jr} \right\} \qquad \text{all } \underline{n}.$$

Consider the right-hand side:

$$\sum_{\substack{i,j \\ r,s}} \frac{q_{ir}^{js}\, I_{ir}}{S_{ir}(n_{ir})} + \sum_{i,r} \frac{q_{ir}^{or}\, I_{ir}}{S_{ir}(n_{ir})} = \sum_{i=1}^{M} \sum_{j=o}^{M} \sum_{r=1}^{R} \sum_{s=1}^{R} \frac{q_{ir}^{js}\, I_{ir}}{S_{ir}(n_{ir})}$$

$$= \sum_{i=1}^{M} \sum_{r=1}^{R} \frac{I_{ir}}{S_{ir}(n_{ir})} \cdot \sum_{j=o}^{M} \sum_{s=1}^{R} q_{ir}^{js}$$

$$= \sum_{i=1}^{M} \sum_{r=1}^{R} \frac{I_{ir}}{S_{ir}(n_{ir})}$$

and

$$\sum_{j,r} X_{or}\, q_{or}^{jr} = \sum_{r} X_{or} \sum_{j=1}^{M} q_{or}^{jr} = \sum_{r} X_{or} = X_o$$

The balance equations therefore reduce to

$$\sum_{\substack{i,j \\ r,s}} p(\underline{n}_{ir}^{js}) \frac{q_{ir}^{js}\, I_{js}}{S_{ir}(n_{ir}+1)} + \sum_{i,r} p(\underline{n}_{ir}^{or}) \frac{q_{ir}^{or}}{S_{ir}(n_{ir}+1)} + \sum_{j,r} p(\underline{n}_{or}^{jr})\, X_{or}\, q_{or}^{jr}\, I_{jr}$$

$$= p(\underline{n}) \left\{ \sum_{i,r} \frac{I_{ir}}{S_{ir}(n_{ir})} + X_o \right\} \qquad \text{all } \underline{n}.$$

2.5 SOLVING THE BALANCE EQUATIONS

The solution of the balance equations is

$$p(\underline{n}) = \frac{1}{G} \prod_{i=1}^{M} \prod_{r=1}^{R} F_{ir}(n_{ir})$$

where

$$F_{ir}(n_{ir}) = \begin{cases} 1 \text{ if } n_{ir} = 0 \\ X_{ir}\, S_{ir}(n_{ir})\, F_{ir}(n_{ir}-1) \quad \text{if } n_{ir} > 0 \end{cases}$$

i.e. $$F_{ir}(n_{ir}) = X_{ir}^{n_{ir}} \prod_{k=1}^{n_{ir}} S_{ir}(k) \quad , \quad n_{ir} > 0.$$

G is a normalizing constant, given by

$$G = \sum_{\underline{n}} \prod_{i=1}^{M} \prod_{r=1}^{R} F_{ir}(n_{ir}) \quad , \text{ where the summation extends over all possible } \underline{n}.$$

Noting that

$$p(\underline{n}_{ir}^{js}) = \frac{X_{ir}\, S_{ir}(n_{ir}+1)}{X_{js}\, S_{js}(n_{js})} \cdot p(\underline{n})$$

$$p(\underline{n}_{ir}^{or}) = X_{ir}\, S_{ir}(n_{ir}+1) \cdot p(\underline{n})$$

$$p(\underline{n}_{or}^{jr}) = \frac{1}{X_{jr}\, S_{jr}(n_{jr})} \cdot p(\underline{n})$$

the above solution is readily verified provided that the X_{ir} satisfy the job flow balance equations. For closed systems, as we have seen, these equations do not allow a unique solution. The analyst can, however, obtain a unique set of visit ratio data and derive the X_{ir} by means of an arbitrary normalization. Note that the routing frequencies q_{ir}^{js} enter purely on the conceptual level and need not be specified: properly measured visit ratios will automatically satisfy the visit ratio equations.

For the purpose of general exposition, we have worked with stratified data and obtained the general solution of the homogenized balance equations. In the remainder of this paper we consider only load-independent service times and shall assume that the network is closed, i.e. the total number of customers is fixed. The load independent service functions are denoted by S_{ir} which is the notation used in Section 2.1 for the mean service time. In practice the analyst might

have available only the mean service time parameters and will use them instead of the load independent service functions. We discuss this aspect further in Section 5. In the next section we

- develop an algorithm to calculate the normalizing constant G;
- discuss the calculation of network- and queue-descriptive measures and develop appropriate algorithms.

3. CALCULATION OF THE NORMALIZING CONSTANT

The normalizing constant is defined as

$$G(N,M,R) = \sum_{\underline{n} \in S(N,M,R)} \prod_{i=1}^{M} \prod_{j=1}^{R} (X_{ij} S_{ij})^{n_{ij}} .$$

where $S(N,M,R) = \{\underline{n} = (n_{11},n_{12},\ldots,n_{1R},n_{21},\ldots,n_{2R},\ldots,n_{MR}) \mid \sum_{i=1}^{M} \sum_{j=1}^{R} n_{ij} = N \;\&\; n_{ij} \geq 0 \;\forall\, i,j\}$.

Generalizing the approach followed by Buzen [BUZE73], we consider the following function

$$g(n,m,r) = \sum_{\underline{n} \in S(n,m,r)} \prod_{i=1}^{m} \prod_{j=1}^{r} (X_{ij} S_{ij})^{n_{ij}} .$$

Then, for $m > 1$ it follows that

$$g(n,m,r) = \sum_{p=0}^{n} \sum_{\underline{v} \in S(p,r)} \prod_{j=1}^{r} (X_{mj} S_{mj})^{v_j} \; g(n-p,m-1,r)$$

where $S(p,r) = \{\underline{v} = (v_1,v_2,\ldots,v_r) \mid \sum_{j=1}^{r} v_j = p, v_j \geq 0 \;\forall\, j\}$.

Let $h^{(m)}(p,r) = \sum_{\underline{v} \in S(p,r)} \prod_{j=1}^{r} (X_{mj} S_{mj})^{v_j}$.

Then it follows (see [BUZE73]) that

$$h^{(m)}(p,r) = h^{(m)}(p,r-1) + (X_{mr} S_{mr})\, h^{(m)}(p-1,r)$$

with $h^{(m)}(p,1) = (X_{m1} S_{m1})^{p} \qquad p = 0,1,\ldots,N; \;\forall m$

and $h^{(m)}(0,r) = 1$ for $r = 1,2,\ldots,R; \;\forall m; \;\; h^{(m)}(0,0) = 1, \; h^{(m)}(p,0) = 0, \; p \geq 1.$

Thus $g(n,m,r) = \sum_{p=0}^{n} h^{(m)}(p,r)\, g(n-p,m-1,r)$

and the iterative calculation of G(N,M,R) is completed if we observe that

$$g(n,1,q) = \sum_{\underline{v} \in S(n,q)} \prod_{j=1}^{q} (X_{1j} S_{1j})^{v_j} \qquad \text{for any } q \geqslant 1$$

so that $g(n,1,q) = g(n,1,q-1) + (X_{1q} S_{1q}) g(n-1,1,q)$

$$\text{with } g(n,1,1) = (X_{11} S_{11})^{n}, \qquad n = 1,2,\ldots,N$$

$$g(0,1,q) = 1 \; \forall q \; .$$

In fact, this last calculation is unnecessary since

$$g(n,1,q) = h^{(1)}(n,q) \; , \qquad q = 1,2,\ldots,R \quad n = 1,2,\ldots,N \; .$$

Note that in order to calculate the normalizing constant G(N,M,R) we need only calculate $h^{(m)}(p,r)$, $m = 1,\ldots,M$; $r = 1,\ldots,R$ and $p = 1,\ldots,N$; $g(n,m,R)$, $n = 1,\ldots,N$; $m = 2,\ldots,M$; with $g(n,1,R) = h^{(1)}(n,R)$, $n = 0,1,\ldots,N$.

Having calculated the normalizing constant, let us now consider the calculation of various network-descriptive measures.

4. NETWORK-DESCRIPTIVE MEASURES

4.1 UTILIZATION

The utilization of centre i_o by customers of class j_o is defined as follows:

$$U_{i_o j_o} = \sum_{\substack{\underline{n} \in S(N,M,R) \\ \& \; n_{i_o j_o} \geqslant 1}} p(\underline{n}) \cdot \frac{n_{i_o j_o}}{n_{i_o}} \; .$$

If we let $i_o = M$, $j_o = R$, it follows that

$$U_{MR} = \frac{1}{G(N,M,R)} \sum_{\substack{\underline{n} \in S(N,M,R) \\ \& \; n_{MR} \geqslant 1}} \left(\prod_{i=1}^{M} \prod_{j=1}^{R} (X_{ij} S_{ij})^{n_{ij}} \right) \cdot \frac{n_{MR}}{n_M}$$

$$= \frac{1}{G(N,M,R)} \cdot \sum_{m=1}^{N} m \sum_{\substack{\underline{n} \in S(N,M,R) \\ \& \; n_{MR=m}}} \left(\prod_{i=1}^{M} \prod_{j=1}^{R} (X_{ij} S_{ij})^{n_{ij}} \right) \cdot \frac{1}{n_M}$$

$$= \frac{1}{G(N,M,R)} \cdot \sum_{m=1}^{N} m(X_{MR}\, S_{MR})^m \sum_{p=o}^{N-m} \frac{1}{p+m} \Big(\sum_{\underline{v} \in S(p,R-1)} \prod_{j=1}^{R-1} (X_{Mj}\, S_{Mj})^{v_j} \Big)$$

$$\cdot \Big(\sum_{\underline{n} \in S(N-m-p,M-1,R)} \prod_{i=1}^{M-1} \prod_{j=1}^{R} (X_{ij}\, S_{ij})^{n_{ij}} \Big)$$

$$= \frac{1}{G(N,M,R)} \cdot \sum_{m=1}^{N} m(X_{MR}\, S_{MR})^m \sum_{p=o}^{N-m} \frac{1}{p+m}\, h^{(M)}(p,R-1)\, g(N-m-p,M-1,R) .$$

4.2 AVERAGE QUEUE LENGTHS

The average number of customers of class j_o at centre i_o is defined by

$$\bar{n}_{i_o j_o} = \sum_{\substack{\underline{n} \in S(N,M,R) \\ \& \; n_{i_o j_o} \geqslant 1}} p(\underline{n}) \cdot n_{i_o j_o}$$

and it follows that, for $i_o = M$, $j_o = R$,

$$\bar{n}_{MR} = \frac{1}{G(N,M,R)} \sum_{m=1}^{N} m(X_{MR}\, S_{MR})^m \sum_{p=o}^{N-m} h^{(M)}(p,R-1)\, g(N-m-p,M-1,R) .$$

4.3 AVERAGE REQUEST THROUGHPUT RATE AT CENTRE M

Using the operational relation $U_{ir} = X_{ir}\, S_{ir}$ it follows that, having calculated U_{MR}, $X_{MR} = U_{MR}/S_{MR}$, the request throughput rate for class R customers at centre M.

4.4 AVERAGE RESPONSE TIME AT CENTRE M

By Little's Operational Formula (see Section 2.1)

$$R_{ir} = \frac{\bar{n}_{ir}}{X_{ir}} \quad \text{so that} \quad R_{MR} = \frac{\bar{n}_{MR}}{X_{MR}} .$$

4.5 CALCULATION OF DESCRIPTIVE MEASURES FOR REMAINING CLASSES AND CENTRES

So far we have calculated the main descriptive measures, viz. utilization and average queue length, only at centre M and for class R customers.

The calculations for class r customers at centre M follow directly the lines set out above, after classes have been renumbered to interchange classes R and R-s, s=1,2,...,R-1 successively. Upon completion of the calculations for centre M, the descriptive measures for centres M-1,...,1 are calculated by renumbering the centres to interchange centres M and M-p, p=1,...,M-1 successively and repeating the calculations described above for centre M and class R, R-1,...,1.

5. COMPARISON WITH STOCHASTIC NETWORK MODELS

In [DENN77], Denning and Buzen gave an excellent discussion of the limitations of operation analysis. Comparing the multiclass model presented in this paper with the traditional stochastic multiclass models, the following conclusions can be drawn.

a. In operational network models no stochastic assumptions are needed - assumptions which normally cannot be validated or even substantiated. Only measurable operational data are used and the assumptions made are intuitively appealing. The model is conceptually simpler than traditional multiclass models: the concept of customer class groups is used to describe main job classes and its associated system job classes, in place of the usual concept of ergodic subchains of a Markov chain.

   Although interclass branching probabilities are used in deriving the balance equations, only average service times and request throughputs (or visit ratios) are needed on a per centre, per class basis. In order to use the model, the analyst therefore has to extract only these quantities from the workload/monitor data.

b. No assumption has been or need be made about the distribution of service times at any centre.

c. No explicit assumptions are made concerning the type of service centre that each device represents. In multiclass stochastic models, the analyst has the choice of specifying the type of each centre and can choose between first-come-first-served, processor sharing, infinite server and pre-emptive-resume last-come-first-served service diciplines. We have, however, made serious approximations in expressing the state transition rates in terms of measurable quantities. In expressing the state transition rate

$$H(\underline{n}_{ir}^{js},\underline{n}) \text{ as } \frac{q_{ir}^{js}}{S_{ir}(n_{ir}+1)} \text{ (and similarly for other state transitions) we}$$

made two assumptions. First, that $\frac{Q(\underline{n}_{ir}^{js},\underline{n})}{T(\underline{n}_{ir}^{js})} = \frac{C_{ir}^{js}(n_{ir}+1)}{T_{ir}(n_{ir}+1)}$,

i.e., that the rate of state transitions from $\underline{n}_{ir}^{js}$ to $\underline{n}$ equals the rate at which device i throughputs class r customers, when the number of class r customers equals $n_{ir}+1$, which immediately go to device j as class s customers irrespective of the number of class s customers already at device j. This is analogous to the assumption of 'device homogeneity' of Denning and Buzen [DENN78]. Second, that

. $C_{ir}^{js}(n_{ir}+1)$ equals $q_{ir}^{js}C_{ir}(n_{ir}+1)$. By definition, $q_{ir}^{js} \sum_{n\geq 0} C_{ir}(n) = \sum_{n\geq 0} C_{ir}^{js}(n)$ and we are assuming that $q_{ir}^{js} C_{ir}(n) = C_{ir}^{js}(n)$ for all n, i.e., the routing frequencies are independent of the state of the system (but may depend on the total load N). This is analogous to the assumption of 'routing homogeneity' [DENN78].

d. In Section 2.5 we noted that the analyst would often replace the load dependent service functions $S_{ir}(n)$ by S_{ir}, the overall mean service time for class r customers at device i. When this approximation is used in the calculation formulae we are implicitly assuming that the request completion rate for class r customers at centre i is not influenced by customers of other classes at centre i. In effect, device i behaves as if it consisted of R parallel virtual servers, one for each customer class.

6. CONCLUSION

An operational treatment of multiclass networks was presented and computational algorithms derived for obtaining network descriptive measures. The discussion was limited to networks with one class group. The inclusion of more than one class group presents no problems and results for this more general case can be readily obtained. Computational experience with the model seems to indicate that compared with BCMP results, utilizations are predicted with an absolute error of 2-5% and queue lengths within 5-20%.

7. ACKNOWLEDGEMENT

The author gratefully acknowledges the help of J.C. van Staalduinen who checked the manuscript and provided some valuable comments. He is also indebted to J.P. Buzen for a stimulating discussion concerning the handling of device types.

REFERENCES

BASK75 Baskett, F., Chandy, K.M., Muntz, R.R. and Palacios, F.G., 'Open, Closed and Mixed Networks of Queues with Different Classes of Customers', J ACM 22,2 (1975) 248 - 260.

BUZE73 Buzen, J.P., 'Computational Algorithms for Closed Queueing Networks with Exponential Servers', Comm ACM 16,9 (1973) 527 - 531.

BUZE76 Buzen, J.P., 'Fundamental Operational Laws of Computer System Performance', Acta Informatica 7,2 (1976) 167 - 182.

DENN77 Denning, P.J. and Buzen, J.P., 'Operational Analysis of Queueing Networks', Proceedings of the Third International Symposium on Modelling and Performance Evaluation of Computer Systems, Bonn, October, 1977, (North-Holland, Amsterdam, 1977) 151 - 172.

DENN78 Denning, P.J. and Buzen, J.P., 'The Operational Analysis of Queueing Networks', preprint of a paper to appear in Computing Surveys, September, 1978.

GORD67 Gordon, W.J. and Newell, G.F., 'Closed Queueing Systems with Exponential Servers', Operations Research 15 (1967) 245 - 265.

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

HOMOGENEOUS APPROXIMATIONS OF GENERAL QUEUEING NETWORKS[1]

Gianfranco Balbo[2] and Peter J. Denning

Department of Computer Sciences
Purdue University
West Lafayette, IND. 47907
USA

Abstract: Product form queueing networks are of special interest because the algorithms for computing their performance statistics are fast. The concept of homogeneity, which arises in the operational analysis of queueing network models, asserts that the product form is exact if each device's on-line service function is the same as would be observed off-line under constant loads. (A device's service function gives the mean time between departures conditioned on the queue length.) We will show that, corresponding to any given general queueing network, there exists a product-form queueing network of the same topology whose queue length distributions are identical to those of the given network; this suggests that errors in approximations based on product-form models can be confined to parameter estimation. A numerical study compares several practical methods of approximating the service functions from on-line measurements.

[1] Work reported herein was supported in part by NSF Grants GJ-41289 and MCS78-01729 at Purdue University.

[2] Presently on leave from Istituto di Scienze dell'Informazione, Universita' di Torino, ITALY.

1. INTRODUCTION

Queueing network models in which the solution for state occupancies p(n) [the proportion of time the system spends in state n] are of the product form lead to very fast algorithms for computing performance metrics [BALB77, BUZE73, DENN78, REIS75, REIS78]. Stochastic queueing network models satisfying the BCMP theorem [BASK75] are of this kind. So are homogeneous operational queueing networks, in which the on-line behavior of every device is the same as its off-line behavior [DENN77, DENN78]. The fast algorithms make product-form queueing network models very useful even when all the assumptions are not exact.

Exact solutions for p(n) in nonhomogeneous systems are computationally slow. In principle, one can use Cox's method to approximate each device, with arbitrary precision, as a group of "stages". One can then write a set of balance equations among quantities like p(n,m), where n is an apportionment of the jobs among the devices and m is a vector specifying the stage of service currently in progress at each device. Gauss-Seidel elimination, one of the most efficient solution techniques known for such equations, often runs for a considerable time before converging on a solution, especially if the coefficient of variation between departures for some device is high.

To avail themselves of fast solution techniques, analysts prefer to work with a product-form model that approximates the real system. A number of successful approximations have been discovered, but little is known in general about their errors.

The first result of this paper is that there exists a product-form queueing network model in which the marginal queueing distribution of each device is identical to that of a given arbitrary general queueing network. We call this the homogeneous equivalent model (HEM) of the given general queueing network. This result suggests that the product-form model is not the fundamental limitation of approximations; parameter estimation is the limit. The paper also describes a numerical study comparing three methods of approximating the equivalent model: load independent devices, load dependent devices whose service functions are the ones measured on-line in the real system, and the extended product form (EPF).

The experimental part of this investigation complements and extends prior work by other authors -- e.g., CHAN75, CHAN78, GELE75, GELE76, SAUE75, SEVC77, and SHUM76. We omitted the diffusion approximation [GELE75, GELE76, KOBA74] because it is outside the context of product-form queueing network models. We omitted approximations based on the Chandy-Herzog-Woo theorem, which already have been well studied elsewhere [CHAN75, CHAN78, SAUE75]. We also omitted Kuhn's method of accounting for nonexponential devices because it too is not based on product-form queueing networks [KUHN76]. The contributions of this experimental study are a) a comparison of the effects of the coefficient of variation (CV) on various approximations for utilizations and mean queue lengths; b) an evaluation of the relative importance of backlogs caused by bottlenecks or by high CVs; and c) a study of on-line and off-line behavior of devices.

2. FORMAL PROBLEM STATEMENT

A product-form closed queueing network with N jobs and M devices has states $\underline{n} = (n_1, n_2, \ldots, n_M)$ where $N = n_1+n_2+\cdots+n_M$. The proportion of time state $\underline{n}$ is occupied is given by the product form

$$p(\underline{n}) = \frac{1}{G} \prod_{i=1}^{M} F_i(n_i)$$

where G is a normalization constant and F_i is a "device factor":

$$F_i(n) = V_i^n \; S_i(n)S_i(n-1) \cdots S_i(1)$$

where V_i, the visit ratio, is the mean number of visits to device i per job, and $S_i(n)$, the service function of device i, is the mean time between completions conditioned on the queue length being n.[1] If device i is load independent with mean time between completions S_i, its device factor is $F_i(n) = (V_iS_i)^n$. The queueing distribution at device i is defined as

$$p_i(n) = \sum_{\underline{n},\ n_i=n} p(\underline{n}) \ .$$

The question of primary interest here is: Does there exist a choice of service functions $\{S_i(\cdot)\}$ for which the $p_i(n)$ calculated in the model are identical to the $p_i(n)$ observed in the real system? Such service functions define the equivalent homogeneous devices corresponding to the real devices. We will exhibit a fast, iterative method for computing these equivalent functions. We will then compare approximations in terms of the $\{S_i(\cdot)\}$ they generate.

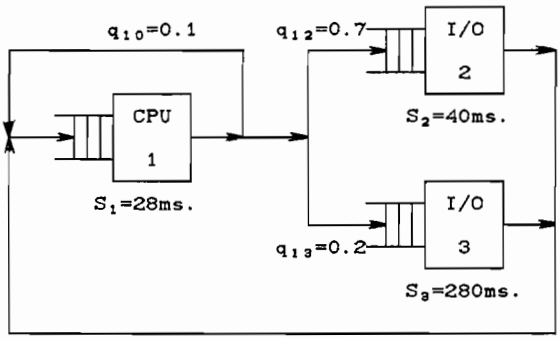


Figure 1. Example network

[1] The service function of device i is defined operationally as the ratio $S_i(n) = T_i(n)/C_i(n)$, where $T_i(n)$ is the total time during which device i is observed to have a queue of length n and $C_i(n)$ is the total number of service completions observed when the queue length is n. Note that $1/S_i(n)$ is the device's output rate observed relative to time intervals in which queue length is n. Note also that $S_i(n)$ need not be constant even if the device contains a single, load independent server [DENN78].

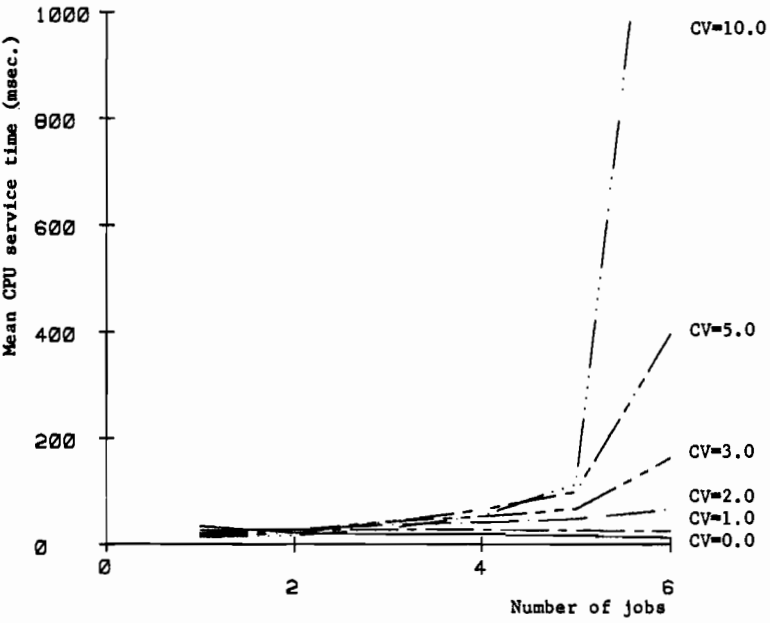


Figure 2. On-line service function for CPU in example network.

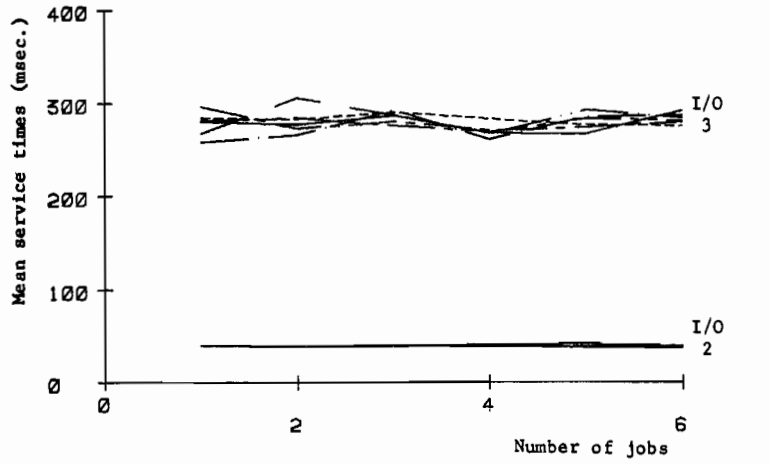


Figure 3. On-line service functions of I/O devices in the example network for various CVs of CPU.

3. METHODOLOGY OF THE INVESTIGATION

To illustrate the homogeneous equivalent service functions and their approximations, we used a simulator of a central server network (Figure 1, with one CPU and two I/O devices) to generate behavior sequences. These sequences were interpreted as observations of a real system. The network is closed with N jobs in it for N=6. (No significant changes in the results were observed for N>6.) Except when we note explicitly otherwise, the simulations used exponential service distributions (CV=1) at the two I/O devices and either hyperexponential service (CV>1) or Erlangian service (CV<1) at the CPU. All three devices were single servers.

Busy times and completion counts for each device at each queue length ($0 \leq n_i \leq N$) were measured from the observed behavior sequence and used to calculate the visit ratios $\{V_i\}$, the queue length distributions $\{p_i(n)\}$, the on-line service functions $\{S_i(n)\}$, the utilizations $\{U_i\}$, and the mean queue lengths $\{\bar{n}_i\}$. These measurements were repeated for a variety of combinations of the CVs of the three devices. This allowed us to study the effect of several high CVs on the errors arising from various approximations to the homogeneous equivalent service functions.

The box below summarizes the three product form approximations which will be compared later in the paper.

CODE	DESCRIPTION
HEM	Homogeneous model equivalent to real system.
HLI	Homogeneous Load Independent Model; the mean service times of devices match those measured on the real system.
HLD	Homogeneous Load Dependent Model; the service functions match the on-line service functions observed in the real system.
EPF	Extended Product Form model for a network of load independent servers; the mean and CV of the service distributions match those measured on the real system.

4. CAUSES OF NONHOMOGENEOUS BEHAVIOR

The coefficient of variation (CV) of a random variable is the ratio of its standard deviation to its mean. A large amount of variation in the times between service completions at a device will cause the device's on-line service function to differ from its off-line service function.

Consider a single-server device (i) with high CV in its distribution of service requests. When observed off-line under any fixed, nonzero queue length (n), the mean time observed between

completions ($S_i(n)$) will always be the mean (S_i) of the request distribution. But when device i is on-line, there will be long requests that block large numbers of short ones. causing long departure times to be correlated with long queue lengths — i.e., $S_i(n)$ will tend to increase in n.[2]

Figure 2 depicts the on-line service function $S_i(n)$ of the CPU in the network of Figure 1, for various values of the CV of the CPU burst distribution.

This figure confirms the above intuition: the higher the CV the stronger the tendency for $S_i(n)$ to increase with the queue size n. The high CV can, like a bottleneck, generate a backlog at a device. The line corresponding to CV = 1 is close to the off-line service function, which is constant at 28 msec.

For small CV (requests nearly all of the same size) there is only a weak tendency for a long queue to build and, hence, the service function will decrease as queue length increases. Figure 2 shows that this decrease is nowhere near as severe as the increase wrought by high CV. Indeed, our experiments (described later) show that very small CVs do not cause much error in the approximations made by product-form models.

Figure 3 shows the on-line service functions of the two (exponential) I/O devices of Figure 1. The presence of the high CV at the CPU has a marginal influence of no consistent pattern on the service functions of these other devices. This is not surprising since exponential servers are homogeneous.

The high CV in a device's output process causes the on-line service function to differ from the off-line because long request will block large numbers of short requests. It follows that a device with a high degree of internal parallelism can mitigate this effect by providing alternate paths on which short requests can bypass a long one. Three examples of such devices are the infinite server, the processor-sharing server, and the last-come-first-served server of the BCMP Theorem [BASK75]; these kinds of devices exhibit the same service functions on-line as they do off line and, hence, they are homogeneous.

An arbitrary subsystem can be approximately homogeneous relative to its environment if it allows a high level of concurrency among jobs inside it. In this case, long jobs cannot effectively block groups of subsequently arriving short ones and, consequently, the on-line and the off-line behaviors will be similar. A study of the Purdue MACE System confirms this; it revealed that the CV of time between completions of jobs in the same time sharing workload

---

[2] An extreme case will illustrate this intuition. Suppose that the N jobs of a closed system have CV=N-2 at device i. It can be shown that one job must require at least $(N-1)S_i$ sec. at device i and the others collectively require less than S_i sec. there. If, at time t, the long job leaves behind in the queue of device i the N-1 short jobs, then all N jobs will leave device i in the interval $[t,t+S_i]$; if the response time of the rest of the network is not too large, then all the short jobs will return to join the queue behind the long one. In this case the high CV at device i forces the frequent recurrence of the state in which all jobs are present at device i.

was 1.06, whereas the CV in the total CPU requirement was 2.15.

It is tempting to conjecture that using the actual on-line service functions as parameters to the product form solution would cause the product form to be exact. It will be evident from our numerical studies that this is not so. (Thus, the homogeneous equivalent service functions differ from the on-line service functions.)

5. EQUIVALENT HOMOGENEOUS NETWORK OF QUEUES

The objective is to calculate service functions $\{S_i(n)\}$ which, when used in the product form solution, lead to the same values of $\{p_i(n)\}$ as observed in the real system. Each resulting $S_i(n)$ is interpreted as the service function of a homogeneous device equivalent to device i in the real system.

The idea of replacing homogeneous subnetworks with equivalent devices is not new; it is the basis of equivalence and decomposition approaches used by Brandwajn [BRAN74, BRAN77] and by Chandy, Herzog and Woo [CHAN75]. The new aspect of our result is that a homogeneous equivalent for an arbitrary queueing network exists. This result demonstrates that homogeneity is not the inherent limitation of product-form queueing network models. The errors arise in parameter estimation, particularly the queue-dependent service functions of devices.

The definition of the normalizing constant G in the product form expression for $p(\underline{n})$ is

$$G = g(N,M) = \sum_{\underline{n}\in S(N,M)} \prod_{i=1}^{M} F_i(n_i)$$

where $S(N,M)$ is the set of all M-component vectors whose nonnegative elements sum to N, and $F_i(n_i)$ is the factor for device i. The proportion of time during which n_M=n can be written as

$$\begin{aligned} p_M(n) &= \sum_{\substack{\underline{n}\in S(N,M) \\ n_M=n}} p(\underline{n}) \\ &= \frac{F_M(n)}{g(N,M)} \sum_{\underline{n}\in S(N-n,M-1)} \prod_{i=1}^{M-1} F_i(n_i) \\ &= F_M(n) \frac{g(N-n,M-1)}{g(N,M)} \end{aligned}$$

Because $F_M(n) = V_M S_M(n) F_M(n-1)$, this reduces to an expression for the homogeneous equivalent service function of device M:

$$S_M(n) = \frac{1}{V_M} \frac{p_M(n)}{p_M(n-1)} \frac{g(N-n+1,M-1)}{g(N-n,M-1)}, \quad n=1,\ldots,N,$$

This expression can be written in the form of a balance equation

$p_M(n)/S_M(n) = p_M(n-1)\lambda(n-1)$, where

$$\lambda_M(n) = V_M \frac{g(N-n,M-1)}{g(N-n+1,M-1)}$$

is the arrival rate to device M generated by the remainder of the network.

Figure 4 shows the algorithm for calculating the homogeneous equivalent service function. It iteratively improves trial functions until the error between trials functions becomes small. A component of the algorithm is a queueing network evaluation routine,

$$G := QN(\{V_i\},\{S_i(n)\},j),$$

for which $\{V_i\}$ are the visit ratios, $\{S_i(n)\}$ are the trial service functions, and j is a "distinguished device" ($1 \leq j \leq M$). QN reindexes the devices so that j corresponds to the last column of the matrix $g(\cdot,\cdot)$ and then executes the standard algorithm [BUZE73] until the M-1st column of the matrix $g(\cdot,\cdot)$ is filled. QN returns an array G[0,...,N] such that $G[r] = g(r,M-1)$, r=0,...,N.

Steps 3 and 4 in the algorithm iterate until there is no significant further change in the trial $\{S_i(n)\}$. (In the cases we studied, two iterations of Step 3 the model yielded estimates of the metrics of the real system to one significant digit of accuracy; ten iterations of step 3 gave three digits of accuracy.) Step 5 ensures that the throughputs of the equivalent network match those of the real system.

The algorithm works for the following reasons. First, there is a unique set of $\{S_i(n)\}$ whose throughputs match the real system and which generate the original $\{p_i(n)\}$. The existence of these $\{S_i(n)\}$ is guaranteed by the equation used in Step 3.2.1. An examination of the product form solution for p(n) shows that changing any proper subset of the values $\{S_i(n)$: all i and n$\}$ would change the speed of some part of the system relative to others; this would cause relative changes in queueing and hence in the $\{p_i(n)\}$, contradicting the supposition that $\{S_i(n)\}$ are the equivalent functions. Second, the equivalent network has the same topology as the original system. Therefore, its throughputs $\{X_i\}$ obey to the throughput equations of the real system. This means that the throughputs of the model after Step 4 are correct to within a constant. Step 5 scales all the service functions to cause the model's $\{X_i\}$ to match those of the real system. (Scaling all the $\{S_i(n)\}$ by the same factor does not change the $\{p_i(n)\}$.) Third, convergence is assured because, in closed networks, the global effect of changing a parameter is less than the change itself [WILL76].

Figure 5 compares the homogeneous equivalent service functions with the actual on-line service functions of the CPU in the network of Figure 1. Notice that the homogeneous equivalents are relatively flat for $0 \leq n \leq N-1$ with a sharp rise for n=N; the service-time at n=N is higher for larger CVs. Figure 6 shows the homogeneous equivalents for the two I/O devices in Figure 1; even though they are exponential, their equivalent functions depend on the load.

INPUT: N, M, T, {C(i)}, {V(i)}, {$p_i(n)$} } i=1,...,M .

OUTPUT: {$S_i(n)$} } n=1,...,N .

ALGORITHM:

1. As an initial trial service function for each i, use the overall mean time between completions (T is the total observation time):

$$S_i(n) = \frac{T - T_i(0)}{C(i)} \qquad n=1,\ldots,N.$$

2. Initialize error measure: E := 0.

3. For j=1,...M do: (for each device)

   3.1. Calculate G := QN({V_i},{$S_i(n)$},j)

   3.2. For n=1,...,N do: (for each queue length)

      3.2.1. Calculate new trial service function value

$$Y = \frac{p_j(n)}{p_j(n-1) * V_j} \quad \frac{G[N-n+1]}{G[N-n]} .$$

      3.2.2. Aggregate the squared error

$$E := E + \left[\frac{S_j(n)-Y}{S_j(n)}\right]^2 .$$

      3.2.3. Update the service function

$$S_j(n) := Y .$$

4. If E > desired ε, repeat from Step 3.

5. (Ensure throughput constraint.)
   Let $a = \frac{1}{X(1)} \sum_{n=1}^{N} p_i(n)/S_i(n)$, where X(1) = C(1)/T.
   Scale by replacing each $S_i(n)$ with $S_i(n)/a$.

Figure 4.

6. EXPERIMENTAL STUDIES

Because there is no known direct measurement of the real system that will yield the homogeneous equivalent service functions, we conducted a series of experiments to see how well models based on easily measured quantities approximate the equivalent service functions. The easiest approximation replaces each function $S_i(n)$ with the overall mean service time, $(T-T_i(0))/C_i$; we called this the homogeneous load independent (HLI) model . A second approximation puts each $S_i(n)$ equal to the actual on-line service function; we called this the homogeneous load dependent (HLD) model. We also studied the extended product form (EPF) of Shum and Buzen, which has given good results elsewhere [SHUM76, SHUM77]; the EPF incorporates explicitly the CV of each device's service distribution. The three approximations were evaluated by comparing calculated utilizations and msan queue lengths of devices with the corresponding values measured in the behavior sequences of the simulated system.

6.1. HLI Approximation

In this series of experiments we used the simulator to generate behavior sequences for values of the CPU's CV ranging from 0 to 10. For each sequence and for each device we measured the overall mean times between completions $\{S_i\}$ and the visit ratios $\{V_i\}$, which we then used as the parameters for the product-form load-independent queueing network evaluator.

Figure 7 compares the actual utilization of the CPU with the values obtained from the HLI model, and displays the relative error between the HLI estimate and the actual value. Since the the same $\{V_i\}$ and $\{S_i\}$ were used in all the behavior sequences,the utilization ratios are independent of the CV [DENN78]:

$$\frac{U_i}{U_j} = \frac{V_i S_i}{V_j S_j} .$$

Therefore the utilizations of the I/O devices are scaled versions of the CPU utilization: the relative error between the HLI approximation and actual utilization is the same for all devices.

The HLI model consistently overestimates the utilizations when the CPU's CV > 1. This is because, first, the utilizations are constrained to be in fixed ratios, and, second, the high CV causes jobs to backlog at the CPU which in turn lowers the utilization of the I/O devices.

For utilizations, the HLI model has very small error when $0 \leq CV \leq 2$, but its errors may exceed 10% as soon as $CV \geq 6$. A separate series of experiments with all the $CV \approx 0$ also showed that the model's estimates of utilizations are correct within 3%.

Figure 8 compares the means of the queue lengths estimated by the HLI model with the actual means. The errors are zero when CV = 1 because, in the simulations used to generate behavior sequences, the CPU's service distribution was exponential (and therefore homogeneous) when CV = 1. The HLI model is much less robust in estimating queue lengths than it is in estimating utilizations; in this case, for example, its estimates of $\bar{n}_i$ are accurate to within

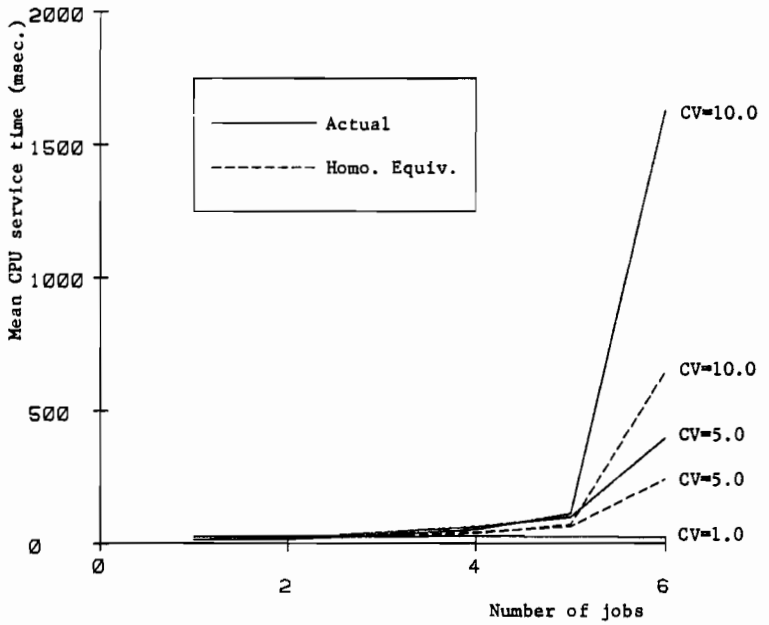


Figure 5. Comparison between actual on-line service function of the CPU and the homogeneous equivalent service function.

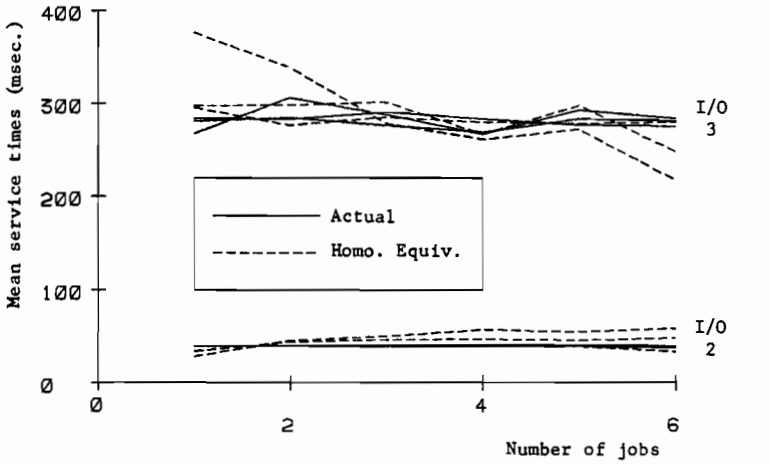


Figure 6. Homogeneous equivalent service functions for I/O devices for various CVs of CPU.

10% only for $0.5 \leq CV \leq 1.8$.

As noted in Section 4, the high CV can generate a backlog at a device. But the only kind of backlog representable in a load-independent model is the standard bottleneck, such as device 3 in our example. The high CV transfers some of the backlog from device 3 (I/O) to device 1 (CPU). This explains why the HLI model overestimates the queue at device 3 and underestimates it at device 1. (Since device 2 causes no backlogs, the HLI model estimated its queue length well.)

To study how CV-induced backlogs can compete with or reinforce bottleneck-induced backlogs, we ran a series of experiments using progressively higher values of mean CPU time (S_1) with the CPU's CV held fixed at 5.0. The results are shown in Figures 9 and 10.

Figure 9 shows, as in Figure 7, that the HLI model consistently overestimates utilizations no matter what device is the bottleneck. Figure 10 shows, as was noted above, that the HLI model model consistently overestimates the queue length at the bottleneck, even if the bottleneck is also a device of high CV. In our example, S_1 = 56 msec causes balance between the CPU and the I/O device; at this point $U_1 = U_3$ and the two devices are of equal importance as bottlenecks. For $S_1 < 56$ msec, device 3 (I/O) is the bottleneck while for $S_1 > 56$ msec device 1 is the bottleneck.

Of special interest in Figure 10 is that the HLI model is nearly exact at the point where the two dominant devices of the system are balanced ($U_1 = U_3$). This is consistent with a suggestion by Courtois [COUR77, pp 83 ff], who argued that balanced devices tend to be more decomposable than unbalanced ones -- thus conforming more closely to the homogeneity assumption.[3] However, the HLI model gives about 15% overestimate in the CPU utilization at the balance point. The relations among bottlenecks, decomposability, and the accuracy of the HLI model are, evidently, interesting subjects of further research.

6.2. HLD model Approximation

Figure 5 showed large difference between the CPU's homogeneous equivalent service function and the actual service function measured on-line. Our experimental study revealed that these differences cause significant errors in estimating utilizations and mean queue lengths by the load-dependent model (LDM).

Because a high CV at the CPU can cause a backlog there, we expect that system states

$$(CPU, I/O, I/O) = (n_1, n_2, n_3) = (N, 0, 0)$$

account for progressively larger proportions of state occupancy when the CPU CV increases. We found that, indeed, p(N,0,0) grows with the CV, but that the HLD model seriously overestimates p(N,0,0). Figure 11 confirms the tendency of the HLD model to overestimate the

[3] It is also interesting that the M/M/1 formula for mean queue length, $\bar{n} = U/(1-U)$, is nearly exact at the balance point, where U is the actual utilization of the device.

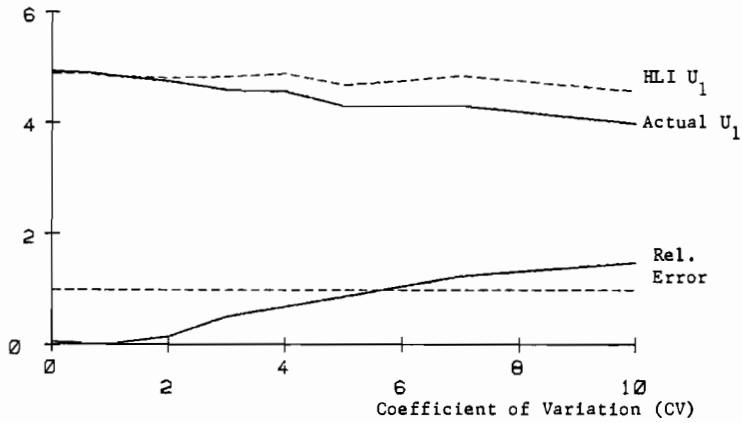


Figure 7. CPU utilization: Actual values, HLI estimates, and relative errors.

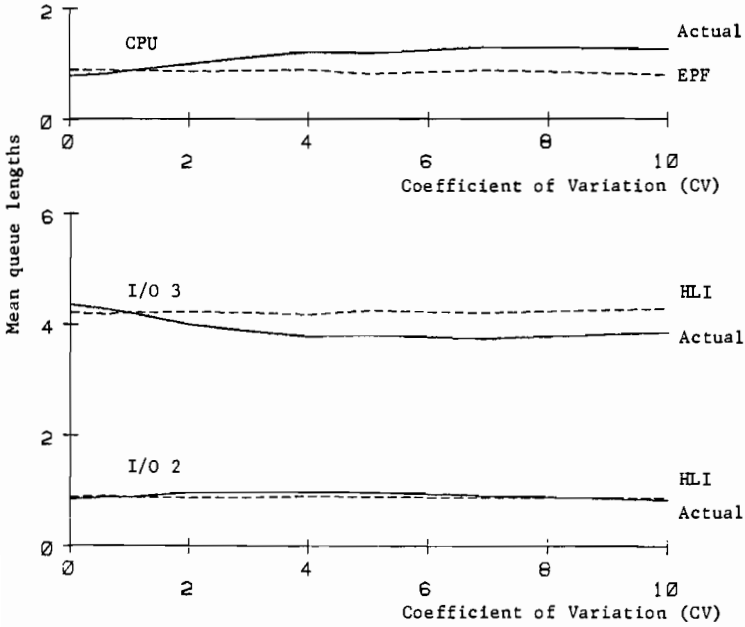


Figure 8. Mean queue lengths in HLI model.

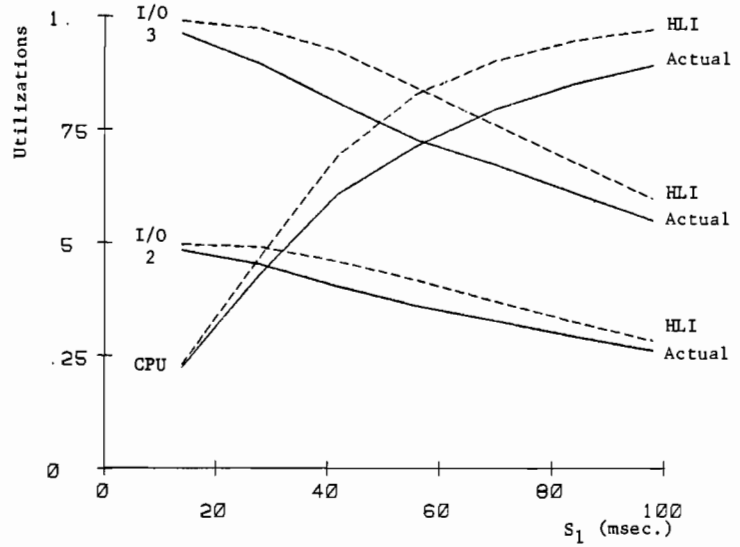


Figure 9. Effect of location of bottleneck on utilization estimates in HLI model.

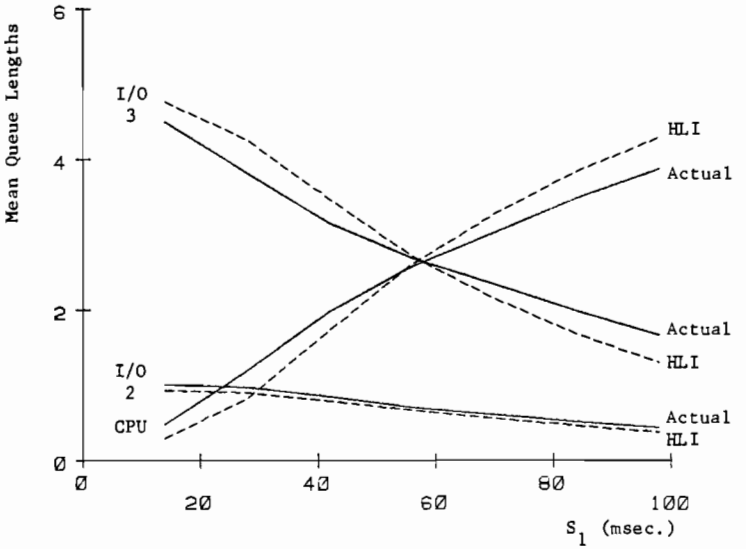


Figure 10. Effect of location of bottleneck on mean queue length estimates in HLI model.

backlog at the device of high CV. The states are ordered according to decreasing likelihood of being observed in the system with N = 6 and CV = 5. The state actually occupied the largest portion of time is (0,0,6) -- reflecting that device 3 is the bottleneck. The state (6,0,0) is actually occupied only about 1/3 as often as (0,0,6) -- showing that the bottleneck is more important than the backlog caused by high CV. In contrast, the HLD model also estimates that states (5,0,1) and (5,1,0), corresponding also to CV-induced backlogs, are more likely than in actuality.

The conclusion is that the HLD model, which uses the on-line service functions attaches far too much importance to backlogs induced by high CV. Backlogs caused by bottlenecks are more important. Because the on-line service function has a shape similar to the homogeneous equivalent service function (See Figure 5), it is possible that a scaling transformation could construct a good approximation to the homogeneous equivalent starting from the on-line function. However we have not investigated this possibility.

Other experiments showed that the HLD model estimated utilizations with almost no error as long as $0 \leq CV \leq 2$. However the utilization errors rapidly multiplied for larger CVs, reaching 10% at CV=3 and 40% at CV=10. The HLD model estimated CPU mean queue length to within 10% only for $0 \leq CV \leq 1.5$. Only when the CPU was the bottleneck did the HLD model give accurate results. The overall conclusion is that the HLD model is less robust than the simpler, HLI model.

6.3. EPF Approximation

In 1976 Shum and Buzen reported an approximation called the extended product form (EPF) [SHUM76, SHUM77]. They noted that the device-factors $\{F_i(n)\}$ in the product form expression for $p(n)$ are proportional to the queue length distributions $\{p_i(n)\}$ in an M/M/1/N queueing system. This insight suggested instead substituting for the $\{F_i(n)\}$ the solution of an M/G/1/N queueing system. By thus incorporating the CVs of the service distributions of the devices into the calculations, this would increase the accuracy of the approximations. A few trial cases suggested that the EPF approximation could estimate actual utilizations and mean queue lengths to within 5% even for CV = 10.

We constructed a version of the EPF algorithm, It is possible to derive service functions $\{S_i(n)\}$ as they would be in an M/G/1/N queueing system, thereby viewing the EPF approximation as another method of estimating the homogeneous equivalent service functions. However, it being more convenient to adopt Shum and Buzen's method intact, we did not explicitly calculate $\{S_i(n)\}$ for the EPF.

As specified by Shum and Buzen, the EPF approximation is computationally difficult to use. The reason is that the $\{S_i(n)\}$ corresponding to the M/G/1/N queueing system depend on the absolute values of the arrival rates $\{\lambda_i\}$ at the devices. Since, in a closed network, the $\{\lambda_i\}$ are not known initially, it is necessary to search the space of all $\{\lambda_i\}$ satisfying the throughput equations of the system until a set $\{\lambda_i\}$ is found for which the completion rate of each device i is also λ_i.

We found instances of the network of Figure 1 in which all the devices had high CVs, and the EPF algorithm could not find a

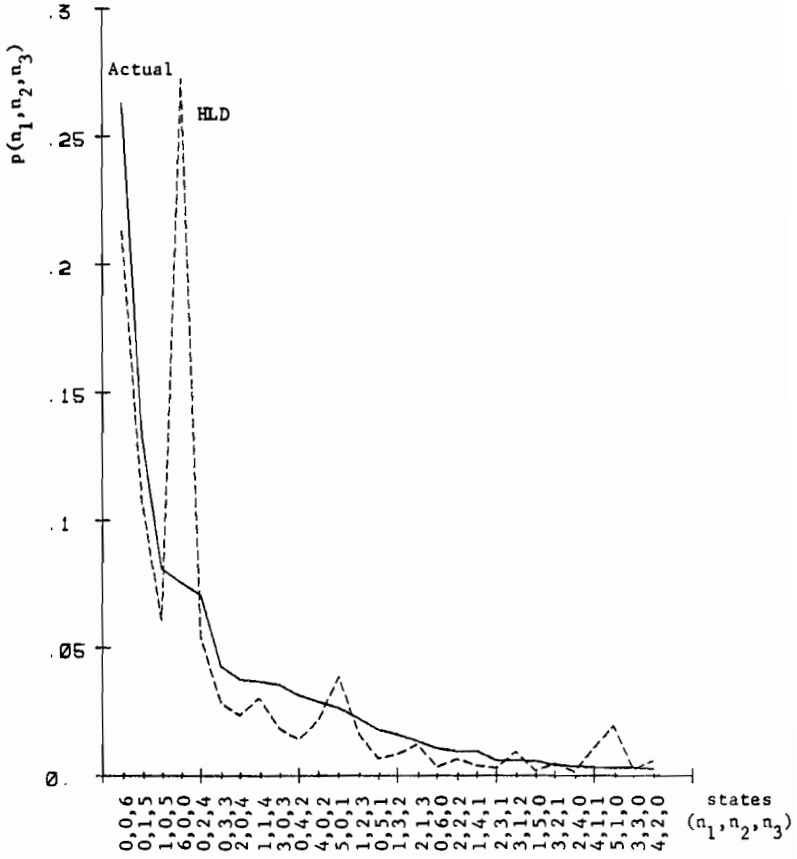


Figure 11. State occupancies.

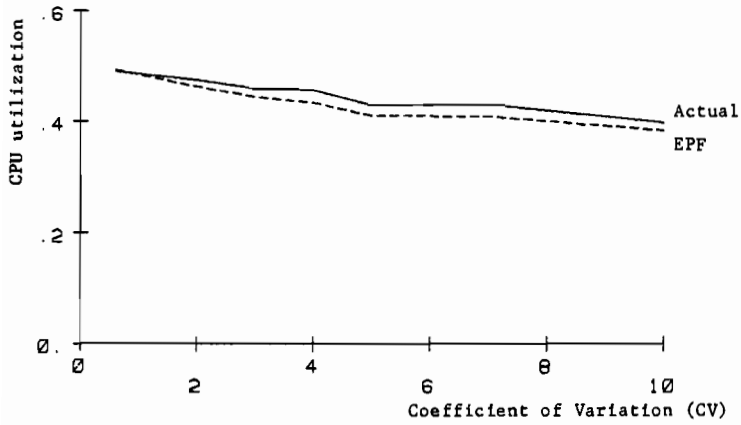


Figure 12. CPU utilization in EPF model.

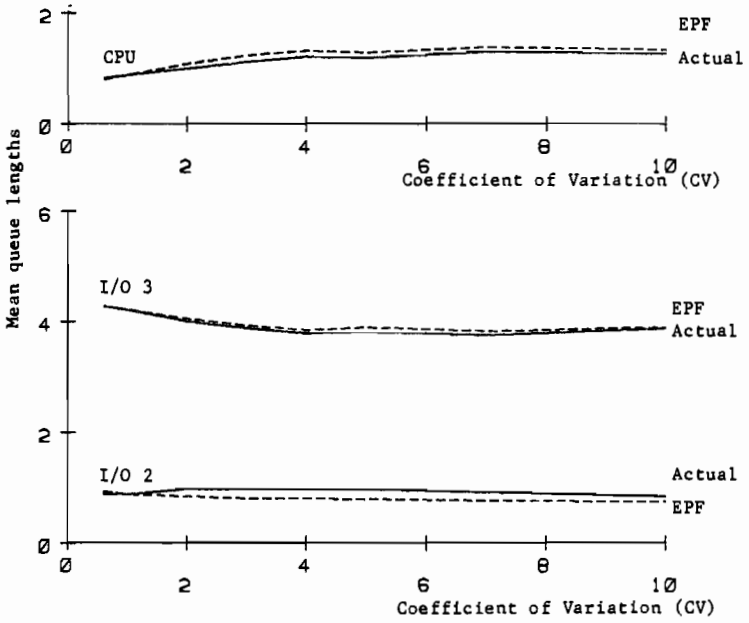


Figure 13. Mean queue lengths in EPF model.

solution. We have since discovered a modified EPF, based on a convex error function, which converges rapidly to a solution in all test cases. (It is the subject of another paper.) The EPF approximation estimates utilizations to within 5% and mean queue lengths to within 10% for all CVs.

Figure 12 illustrates the CPU's utilization (U_1), as estimated by EPF and in actuality, for $0.6 \leq CV \leq 10$. The maximum error is 5%. Similar behaviors were observed for U_2 and U_3, with EPF underestimating consistently and within 5%.

Figure 13 illustrates the mean queue lengths, as estimated by EPF and in actuality, for $0.6 \leq CV \leq 10$. The maximum error in any queue length is 10%, and the error lessens for very high CVs.

The conclusion is that EPF is indeed a robust approximation, but slightly less so than one might expect from reading SHUM76 or SHUM77. The major present limitation is that the published versions of the EPF algorithm may not converge if all devices have high CVs.

7. CONCLUSIONS

We have shown that there exists a product-form queueing network whose marginal queueing distributions are identical to those of a given arbitrary queueing network. A straightforward algorithm rapidly computes the service functions of the equivalent devices when the queueing distributions $\{p_i(n)\}$ are given. This result suggests that the fundamental limitation of queueing network models is not the homogeneity assumption, but rather the inability to estimate the service functions accurately.

The homogeneity assumption asserts that a device's service function measured on-line will be the same as when the service function is measured off-line under constant load. By causing a backlog at a device, a high coefficient of variation (CV) can destroy the homogeneity assumption for that device, producing an error between the product-form model's estimates of utilizations or mean queue lengths and the true values.

Possibly the simplest approximation results from the homogeneous load independent (HLI) model, which sets each $S_i(n)$ to the overall mean service time (S_i) for each value n of the queue length. Our experimental study showed that this model's estimates of utilizations were consistently high and accurate to within 15% for a range on the CPU's CV from 0 to 10. However, this model's estimates of the mean queue length can be significantly in error (e.g. 40%) when the CPU's CV is high. In our study the HLI model was a good estimator ($< 10\%$) of utilization when $0 \leq CV \leq 6$ and for mean queue length when $0.5 \leq CV \leq 1.8$.

The location of the system bottleneck -- at device of high CV or elsewhere -- does not significantly affect the HLI model's (over)estimate of utilization. However, if the devices likely to generate backlogs -- either because of bottlenecks or high CVs -- are in balance (approximately equal utilizations), the HLI model seems to estimate mean queue lengths with very low error. This seems to be consistent with Courtois's argument that balanced systems are more decomposable (and hence more homogeneous) than unbalanced ones. The relations among balance, backlogs, and HLI

model accuracy are worthy of further investigation.

An approximation suggested directly by the definition of homogeneity is the homogeneous load dependent (HLD) model, whose service functions are the ones observed on-line in the real system. This approximation gives much poorer results than the on-line = off-line intuition leads one to expect. The reason is that this model attaches far too much importance to backlogs caused by high CV, and too little importance to backlogs caused by bottlenecks. In reality, "bottleneck backlogs" appear more influential than "CV backlogs". The similarity of shape between the on-line and the homogeneous equivalent service functions suggests that there may exist simple scaling transformations that estimate the latter from the former.

The extended product form (EPF) approximation (in effect) constructs estimates of service functions by using the solution of the M/G/1/N queueing system. This permits both the mean and CV of a device's interdeparture times to be used in the calculation. The drawback of published implementations of the EPF approximation is their searching the space of solutions to the system's throughput equations; this search is slow and may not converge if too many devices have high CVs. When the EPF algorithm does locate a solution, it usually estimates utilizations to within 5% and mean queue lengths to within 10%. Its minimum error occurs for small CVs ($0.8 \leq CV \leq 2$) or very large CVs.

Figures 14 and 15 compare these approximations for the example studied in this paper.

There remains the question of how important the more sophisticated approximations are in practice. In the Purdue Time Sharing Subsystem, for example we observed the CVs shown in Table 1.

Table 1

RANDOM VARIABLE	CV
Times between job submission from all terminals	1.16
Times between job completions by central computing subsystem	1.08
Total CPU requirement per job	2.15

These CVs are sufficiently low that the HLI model can be used with sufficient accuracy. Many subsystems contain sufficient parallelism as to rule out the possibility of observing a high CV in the times between their job-completions.

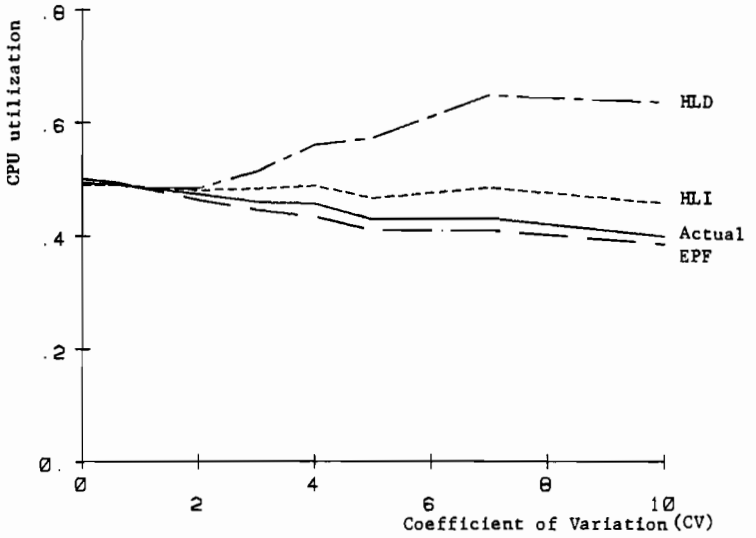


Figure 14. Comparison of CPU utilizations in all models.

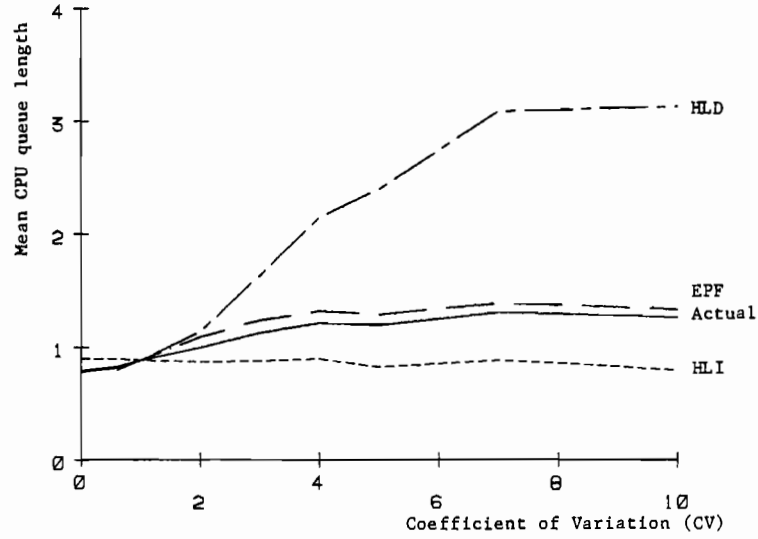


Figure 15. Comparison of mean CPU queue in all models.

ACKNOWLEDGEMENTS

We are grateful to Herbert D. Schwetman for assistance in getting our simulator working; to Steven C. Bruell for providing data on the Purdue System and for helping to implement the EPF algorithm; to Jeffrey Buzen and James Bouhana for insights on the reasons why the high CVs interfere with the predictions of product-form queueing network models.

REFERENCES

BALB77 G. Balbo, S.C. Bruell, and H.D. Schwetman, "Customer Classes and Closed Network Models – A Solution Technique," Proc. 1977 IFIP Congress , Toronto, CANADA, (August 1977), 559-564.

BRAN74 A. Brandwajn, "A Model of a Time Sharing System Solved Using Equivalence and Decomposition Methods," Acta Informatica 4, 1 (1974), 11-47.

BRAN77 A. Brandwajn, "An Approach to the Numerical Solution of Some Queueing Problems," Proc. of the International Symposium on Computer Performance Modeling, Measurement and Evaluation, North-Holland Publishing Co., (August 1977), 83-112.

BUZE73 J. P. Buzen, "Computational Algorithms for Closed Queueing Networks with Exponential Servers," Comm. ACM 16, 9 (September 1973), 527-531.

BUZE76 J. P. Buzen, "Operational Analysis: the Key to the New Generation of Performance Prediction Tools," Proc. IEEE COMPCON, 1976, IEEE, New York.

CHAN75 K.M. Chandy, U. Herzog, and L. Woo, "Approximate Analysis of General Queueing Networks," IBM J. R. & D. 19, (January 1975), 43-49.

CHAN78 K.M. Chandy, and C.H. Sauer, "Approximate Methods for Analyzing Queueing Network Models of Computer Systems," Computing Surveys 10, 3(September 1978).

COOP72 R.B. Cooper, Introduction to Queueing Theory, The MacMillan Company, New York, (1972).

COUR77 P.J. Courtois, Decomposability, ACM Monograph Series, Academic Press (1977).

DENN77 P.J. Denning, and J.P. Buzen, "Operational Analysis of Queueing Networks," Proc. 3rd Int'l Symposium on Modeling and Performance Evaluation of Computer Systems, North-Holland Publishing Co. (1977).

DENN78 P.J. Denning, and J.P. Buzen, "The Operational Analysis of Queueing Network Models," Computing Surveys 10, 3 (September 1978).

GELE75 E. Gelenbe, "On Approximate Computer System Models," J. ACM 22, 2 (April 1975), 261-269.

GELE76 E. Gelenbe, and G. Pujolle, "The Behaviour of a Single Queue in a General Queueing Network," Acta Informatica 7, (1976), 123-136.

KOBA74 H. Kobayashi, "Applications of the Diffusion Approximation to Queueing Networks,"J. ACM 21, 2 (April 1974), 316-338.

KUHN76 P. Kuhn, "Analysis of Complex Queueing Networks by Decomposition," Proc. 8-th International Teletraffic Congress, Melbourne, Australia, (November 1976).

REIS75 M. Reiser, and H. Kobayashi, "Queueing Networks with Multiple Closed Chains: Theory and Computation Algorithms," IBM J. R. & D. 19 (May 1975), 283-294.

REIS78 M. Reiser, and C.H. Sauer, "Queueing Networks Models: Methods of Solution and Their Program Implementations," in Current Trends in Programming Methodology III (K.M. Chandy, and R. Yeh, Eds) Prentice-hall (1978), 115-167.

SAUE75 C.H. Sauer, "Configuration of Computing Systems: an Approach Using Queueing Network Models," Ph.D. Thesis, The University of Texas at Austin, (May 1975).

SEVC77 K.C. Sevcik, A.I. Levy, S.K. Tripathi, and J.L. Zahorjan, "Improving Approximations of Aggregated Queueing Network Subsystems," Proc. of the International Symposium on Computer Performance Modeling, Measurement and Evaluation, North-Holland Publishing Co., (August 1977), 1-22.

SHUM76 A.W. Shum, "Queueing Models for Computer Systems with General Service Time Distributions," Ph.D. Thesis, Division Engrg. & AppliedPhysics, Harvard University, Cambridge, MA 02138 (December 1976).

SHUM77 A.W. Shum, and J.P. Buzen, "The EPF Technique: A Method for Obtaining Approximate Solutions to Closed Queueing Network with General Service Times," Proc. of the 3rd. International Symposium of Modelling and Performance Evaluation of Computer Systems, North-Holland Publishing Co., (October 1977), 201ff.

WILL76 A. C. Williams, andR. A. Bhandiwad, "A Generating Function Approach to Queueing Network Analysis of Multiprogrammed Computers," Networks 6, 1 (1976), 1-22.

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

A STUDY OF FLOWS IN QUEUEING NETWORKS AND AN APPROXIMATE METHOD FOR SOLUTION

G. Pujolle and C. Soula
IRIA-LABORIA
78150 Le Chesnay
France

A new approximate approach to study open single or many server queueing networks is introduced here. The method is based on a theoretical study of the nature of flows to be Poisson or not, in a Jackson network.

INTRODUCTION

The performance of computer systems and data networks are often studied using queueing networks. Unfortunately as soon as the network in somewhat complex, the exact methods no longer exist and we need for accurate approximate methods. Presently, we have three types of such methods : iterative techniques [1] [2], diffusion techniques [3] [4] [5], isolation techniques [6] [7]. Our purpose here is to propose a new approach to study open complex queueing networks with single or many servers queues of the type G/G/s.

As we shall show it in the first section, queueing networks are very complicated to study when a customer can go through a station more than one time. This fact implies that even as in a so simple network as Jackson queueing systems, only some flows possess the Poisson property.

In particular we derive the characteristics of the input, output, feedback and departure processes of exponential queues in series, with a Poisson external arrival process and a global feedback. We show that input process and output process are identically distributed and different of a Poisson process. If n queues are in tandem a recursive formula is given to obtain these distributions.

By a decomposition technique we can predict the property to be Poisson or not of flows in a Jackson network.

As the main approximation in studying queueing networks comes from customers which can go through a station more than one time, we propose a new approach, in avoiding fedback customers. For this, we decompose the network in sub systems characterized by the fact that queues i and j belong to the same sub system if it is possible to go to i from j and to j from i. The only streams which do not possess fedback customers are inter sub network flows.

Then, each station of a sub network is studied in isolation considering that a customer entering the sub network goes through each station at most one time. We obtain mean number in station and response time for each queue, using an appropriate formula obtained by comparisons of available GI/G/1 expressions. The method is extended to take into account stations with many servers. The accuracy of the new method is shown using comparisons with results of simulations on some queueing systems.

M/M/1 QUEUE WITH BERNOULLI FEEDBACK

The model and notations

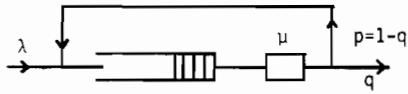


Figure 1

The problem to be studied is illustrated in figure 1.

The arrival process is Poisson with parameter λ. The queueing discipline is FCFS (first come, first served) and the service process is also Poisson with parameter μ. Upon completion of a service, the customer engages a Bernoulli switch : with fixed probability q, he leaves definitively the system, and with probability p=1-q, he rejoins the queue. The feedback is assumed instantaneous. We denote the traffic intensity by $\rho = \frac{\lambda}{\mu q}$.

Service completions will be called "outputs". Exits from the system will be called "departures". Similarly, "inputs" and "arrivals" will be the moments when customers join the queue and enter from outside respectively.

In this paper, we adopt the following notations :

- A(t) : distribution function of the interinput times. Let us introduce the L.S.T (Laplace Stieltjes Transform)
$$\hat{a}(s) = \int_0^\infty e^{-st}\, dA(t)$$

- S(t) : distribution function of the inter departure times

- R(t) : distribution function of the interfeedback times with
$$\hat{r}(s) = \int_0^\infty e^{-st}\, dR(t)$$

- $B(t) = 1 - e^{-\mu t}$ and $\hat{b}(s) = \frac{\mu}{\mu+s}$

The input process

This problem has been studied by P.J. Burke [8].

The distribution function of interarrival times is $1 - e^{-\lambda t}$. Let t_0 be the time of any input, and let G(t) be the probability for the only customer to arrive between times t_0 and t_0+t to be a fedback customer.

Then we can write

$$1 - A(t) = e^{-\lambda t}[1-G(t)] \qquad (1)$$

and if we use the L.S.T; (1) can be transformed to :

$$\hat{a}(s) = \frac{\lambda}{\lambda+s}\,[1 - \hat{g}(\lambda+s)] + \hat{g}(\lambda+s) \qquad (2)$$

if we denote : $\hat{g}(\lambda+s) = \int_0^\infty e^{-(\lambda+s)t}\, dG(t)$

Now, we have to compute $\hat{g}(s)$.

Let $\pi(i)$ the equilibrium probability of state i, just before t_0.

It can be showed (see Burke [8], Disney [9]), that $\pi(i) = \rho^i(1-\rho)$.

Let us denote $\pi(\geq i) = \sum_{j=i}^{\infty} \pi(j) = \rho^i$.

Using the fact that there is no arrival before the next feedback and using the value of $\pi(\geq i)$, we get :

$$\hat{g}(s) = \sum_{i=0}^{\infty} \pi(\geq i)\, pq^i\, [\hat{b}(s)]^{i+1}$$

Therefore

$$\hat{g}(s) = \frac{p\hat{b}(s)}{1-q\hat{b}(s)} \tag{3}$$

and we find :

$$g(t) = p\mu e^{-(\mu-\lambda)t}$$

from (3) we obtain :

$$\hat{g}(\lambda+s) = \frac{p\mu}{\mu+s}$$

Substituting the expression of $\hat{g}(\lambda+s)$ into (2) yields :

$$\hat{a}(s) = \frac{\lambda\mu+s(p\mu-\lambda)}{(\lambda+s)(\mu+s)}$$

or

$$\hat{a}(s) = \frac{p\mu}{\mu-\lambda}\,\frac{\mu}{\mu+s} + \frac{\mu q-\lambda}{\mu-\lambda}\,\frac{\lambda}{\lambda+s} \tag{4}$$

then, if we take the inverse transformation

$$\boxed{A(t) = 1 - \frac{p\mu}{\mu-\lambda}\, e^{-\mu t} - \frac{\mu q-\lambda}{\mu-\lambda}\, e^{-\lambda t}} \tag{5}$$

We have to note that the input process is not Poisson.

Disney and McNickle [9] show this process is not even a renewal process

The departure process

We analyse the departure process by the following substitution : a customer service time is replaced by the addition of the service times of all his successive returns.

Let $B^*(t)$ the p.d.f (probability distribution function) of the equivalent service time. L. Takacs [10] shows the result :

$$B^*(t) = 1 - e^{-\mu q t}$$

therefore, the p.d.f of the interdeparture times is :

$$S(t) = \pi^*(o) \int_0^t (1 - e^{-\lambda(t-x)})\, dB^*(x) + (1 - \pi^*(o))B^*(t)$$

when $\pi^*(i)$ is the stationnary probability of state i, in the equivalent system, but we know (see [8], [9]) that the distribution of the queue size is the same for both systems, then : $\pi(i) = \pi^*(i)$. Finally we get :

$$\boxed{S(t) = 1 - e^{-\lambda t}}$$

As the departure process is renewal [25], it is a Poisson process.

The output process

Let us now, examine the distribution F(t) of the times between two completeness of service. Such an interval is equal to a service time except when the queue becomes empty.
We denote by $\hat{f}(s)$ the L.S.T of the interoutput times.
Therefore, the function $\hat{f}(s)$ is :

$$\hat{f}(s) = \pi'(o)\,\frac{\lambda}{\lambda+s}\,(s) + (1 - \pi'(o))\hat{b}(s)$$

where, $\pi'(o)$ is the probability for the queue to be empty just after an output. And Disney [11] proves the result $\pi'(o) = \pi(o)q = q[1-\frac{\lambda}{\mu q}]$ then we get

$$\boxed{\hat{f}(s) = \frac{p\mu}{\mu-\lambda}\,\frac{\mu}{\mu+s} + \frac{\mu q-\lambda}{\mu-\lambda}\,\frac{\lambda}{\lambda+s}}$$

So, we see that the interinput times and the interoutput times have exactly the same distribution (an hyperexponential distribution). This result was pointed out by R.L. Disney and D.G. McNickle [11]. Moreover, it is shown by Foley [28] this process also fails to be a renewal process.

The feedback process

A complementary result is the L.S.T of the interfeedback times. We know that just after a feedback time the queue is in state i with the stationnary probability $\pi(i-1)$ ([8]). Then, conditionning upon the number of customers in queue we can write

$$\hat{r}(s) = \sum_{i=1}^{\infty} \pi(i-1)\alpha(i)$$

where $\alpha(i)$ is the L.S.T of interfeeback times knowing that the queue is in state i. We use an iterative technique to compute the function $\alpha(i)$. The time of the first event after the feedback is the lower of the first arrival and the first output. So the L.S.T of this time is $\frac{\lambda+\mu}{\lambda+\mu+s}$. With a probability $\frac{\lambda}{\lambda+\mu}$ (resp. $\frac{\mu q}{\lambda+\mu}$) this event is an arrival (resp : a departure) and now the queue is in state i+1 (resp : i-1), and with a probability $\frac{\mu p}{\lambda+\mu}$ this is a feedback. Then we have

$$\alpha(i) = \frac{\lambda+\mu}{\lambda+\mu+s}\,[\frac{\lambda}{\lambda+\mu}\alpha(i+1) + \frac{\mu q}{\lambda+\mu}\,\alpha(i-1) + \frac{\mu p}{\lambda+\mu}] \qquad i > 0$$

$$\alpha(o) = \frac{\lambda}{\lambda+s}\,\alpha(1)$$

This can be seen as an application of the strong Markov property.
We write these equations as follows :

(6) $$(\lambda+\mu+s)\alpha(i) = \lambda\alpha(i+1) + \mu q\alpha(i-1) + \mu p \qquad i > 0$$

(7) $$(\lambda+s)\alpha(o) = \lambda\alpha(1)$$

Using $\pi(i) = \rho\pi(i-1)$ and (7), the summation of (6) for i=1 to ∞ after multiplica-

tion by $\pi(i-1)$ yields :

$$(8) \qquad (\mu p+s)\tilde{r}(s) = \mu p - \frac{\mu p}{\lambda} s\, \pi(o)\alpha(o)$$

Now, we have to compute $\alpha(o)$:

Setting $\alpha(i) = \beta(i) + \frac{\mu p}{\mu p+s}$ in (6), we get the following equation

$$(\lambda+\mu+s)\beta(i) = \lambda\beta(i+1) + \mu q\beta(i-1) \qquad i \geq 1$$

and a solution of this last equation is

$$(9) \qquad \beta(i) = a\rho_1^i$$

where ρ_1, is the lower root of the equation

$$\lambda x^2 - (\lambda+\mu+s)x + \mu q = 0$$

(for more details see [27]).

Now, using (7) and (9), we deduce the system

$$\begin{cases} \alpha(o) = a + \frac{\mu p}{\mu p+s} \\ \frac{\lambda+s}{\lambda}\,\alpha(o) = a\rho_1 + \frac{\mu p}{\mu p+s} \end{cases}$$

So, the L.S.T $\tilde{r}(s)$ is entirely determined. We have

$$(10) \qquad \boxed{(\mu p+s)\tilde{r}(s) = \mu p - \mu q s \pi(o)\, \frac{\mu p\ (1-\rho_1)}{(\mu p+s)[\lambda(1-\rho_1)+s]}}$$

Thus the feedback process is not a Poisson process. This is a new result. However it is not known, if the feedback process is a renewal process.

RESULTS FOR TWO QUEUES IN SERIES

The two tandem queues system with feedback is illustrated on figure 2

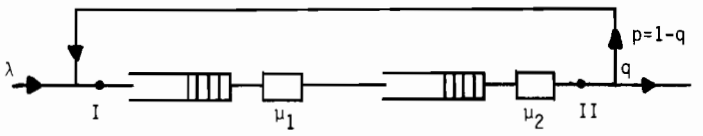


Figure 2

The service times of the first server are distributed exponentially with parameter μ_1 and L.S.T is $b_1(s)$; those of the second server are exponential with paramete μ_2 and L.S.T is $b_2(s)$. A(t) is again the interinput distribution (mark I, on figure 2) and F(t) is the interoutput distribution (mark II, on figure 2), the equality (1) : $1 - A(t) = e^{-\lambda t}[1 - G(t)]$, is still true, whatever the number of tandem queues. In the same way, since the departure process is still a Poisson process with parameter λ, we have symmetrically :

$$1 - F(t) = e^{-\lambda t}[1 - H(t)]$$

Using the L.S.T we obtain

$$\hat{f}(s) = \frac{\lambda}{\lambda+s}\,[1 - \hat{h}(\lambda+s)] + \hat{h}(\lambda+s) \tag{11}$$

where $\hat{h}(\lambda+s) = \int_0^\infty e^{-(\lambda+s)t}\, dH(t)$

And from (11) we get :

$$\hat{h}(s) = \frac{s\hat{f}(s-\lambda)-\lambda}{s-\lambda}$$

Now, we shall see (in part III) that the L.S.T $\hat{f}(s)$ can be easily found for n tandem queues. For n=2, we have :

$$\hat{f}(s) = (1-\rho_1)(1-\rho_2)q\frac{\lambda}{\lambda+s}\hat{b}_1(s)\hat{b}_2(s)+(1-\rho_2)\hat{b}_1(s)\hat{b}_2(s)(p+q\rho_1)+\rho_2\hat{b}_2(s)$$

setting $\rho_i = \frac{\lambda}{\mu_i q}$, (i=1,2).

After some algebra, we find

$$\hat{h}(s) = \frac{p[\mu_1\mu_2 q+\lambda(s-\lambda)]}{q[\mu_1+s-\lambda][\mu_2+s-\lambda]} \tag{12}$$

Let us now compute the function $\hat{g}(s)$. Conditionning upon the states i and j of the queue 1 and 2 we can write :

$$\hat{g}(s) = \sum_{j=0}^{\infty} \pi_2(j) \sum_{i=1}^{\infty} \pi_1(i-1)\,\alpha(i,j) \tag{13}$$

where $\alpha(i,j)$ is the L.S.T of interval of time before the next arrival knowing that the queue 1 is in state i with the stationnary probability $\pi_1(i-1)$ and queue 2 is in state j with the stationnary probability $\pi_2(j)$. Let t_o be the time of any input in queue 1. The first event after t_o is first of both possible service completion, which L.S.T is $\frac{\mu_1+\mu_2}{\mu_1+\mu_2+s}$. Such as for $\alpha(i)$ in I.5 we get the following recurrent equations :

$$(\mu_1+\mu_2+s)\alpha(i,j) = \mu_1\alpha(i-1,j+1) + \mu_2 q\alpha(i,j-1) + \mu_2 p \qquad i > 0,\ j > 0 \tag{14}$$

$$(\mu_1+s)\alpha(i,o) = \mu_1\alpha(i-1,1) \qquad i > 0 \tag{15}$$

$$(\mu_2+s)\alpha(o,j) = \mu_2 q\alpha(o,j-1) + \mu_2 p \qquad j > 1 \tag{16}$$

And of course we have

$$\alpha(o,1) = \frac{\mu_2 p}{\mu_2+s} \tag{17}$$

Let $\gamma(j)$ be defined as : $\gamma(j) = \sum_{i=1}^{\infty} \pi_1(i-1)\alpha(i,j)$

Then the summation of (14) and (15) after multiplication by $\pi_1(i-1)$ yields

$$\begin{cases} (\mu_1+\mu_2+s)\gamma(j) = \mu_1[\pi_1(o)\alpha(o,j+1) + \rho_1\gamma(j+1)] + \mu_2 q\gamma(j-1) + \mu_2 p & j > 0 \\ (\mu_1+s)\gamma(o) = \mu_1[\pi_1(o)\alpha(o,1) + \rho_1\gamma(1)] \end{cases} \tag{18}$$

then, summing (18) upon j, to obtain $\sum_{j=0}^{\infty} \pi_2(j)\gamma(s) = \hat{g}(s)$, we get after reduction :

$$(\mu_1+s-\lambda)\hat{g}(s) = \mu_1 \pi_1(o) \sum_{j=0}^{\infty} \pi_2(j)\alpha(o,j+1) + \mu_2 p\,[1-\pi_2(o)]$$

Now, we need to compute the time $\sum_{j=1}^{\infty} \pi_2(j)\alpha(o,j+1)$. From the equation (16) valid for $j \geq 2$ we get by summation and after reduction and using (17)

$$(\mu_2+s-\lambda) \sum_{j=1}^{\infty} \pi_2(j)\alpha(o,j+1) = \lambda\pi_2(o) \frac{\mu_2 p}{\mu_2+s} + \frac{\lambda p}{q}$$

So, after simplification we finally get

$$\hat{g}(s) = \frac{p[\mu_1\mu_2 q+\lambda(s-\lambda)]}{q[\mu_1+s-\lambda][\mu_2+s-\lambda]}$$

this last expression is strictly equivalent to the one we obtained for the function $\hat{h}(s)$. From this equality we come to the conclusion that the L.S.T $\hat{a}(s)$ and $\hat{f}(s)$ are equal : the interinput times and interoutput times are identically distributed and different of the exponential distribution.

It is interesting (in view of generalization) to note that the last expression that we obtained for $\hat{g}(s)$ can be written as :

$$\hat{g}(s) = p(1-\pi_2(o)) \frac{\hat{b}_2(s)}{1-q\rho_2\hat{b}_2(s)} + p\pi_2(o) \frac{\hat{b}_1(s)\hat{b}_2(s)}{[1-q\rho_1\hat{b}_1(s)][1-q\rho_2\hat{b}_2(s)]}$$

The main conclusion is that the feedback process is not Poisson.

GENERALIZATION TO n TANDEM QUEUES

Such a system is illustrated on figure 3

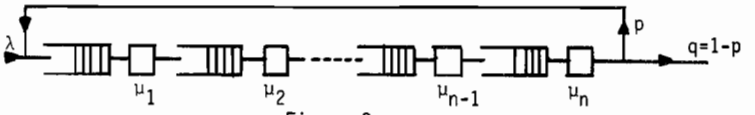


Figure 3

Let $\hat{b}_i(s) = \int_0^{\infty} e^{-st}\, \mu_i e^{-\mu_i t}\, dt \qquad i=1,2,\ldots,n.$

It is convenient to note $\hat{f}(s) = \hat{f}_n(s)$, where n indicate the number of tandem queues.

So we have :

$$\hat{f}_n(s) = q \frac{\lambda}{\lambda+s} \sum_{i=1}^{n} (1-\rho_i)\hat{b}_i(s) + (p+q\rho_1)\hat{b}_1(s) \sum_{i=2}^{n} (1-\rho_i)\hat{b}_i(s) + \sum_{k=2}^{n-1} \rho_k\hat{b}_k(s) \sum_{i=k+1}^{n} (1-\rho_i)\hat{b}_i(s) + \rho_n\hat{b}_n(s) \tag{19}$$

From (19), we deduce the recurrent equation

$$\hat{f}_n(s) = \hat{f}_{n-1}(s)(1-\rho_n)\hat{b}_n(s)+\rho_n\hat{b}_n(s) \quad n=2,3,... \tag{20}$$

Since $\hat{h}_n(s) = \frac{s\hat{f}_n(s-\lambda)-\lambda}{s-\lambda}$, for all n=1,2,.... We show that the functions $\hat{h}_n(s)$ also verify :

$$\hat{h}_n(s) = \hat{h}_{n-1}(s)(1-\rho_n)\hat{b}_n(s-\lambda) + p\rho_n\hat{b}_n(s-\lambda) \tag{21}$$

By setting $\hat{B}_n(s) = \frac{\hat{b}_n(s)}{1-q\rho_n\hat{b}_n(s)} = \frac{\mu_n}{\mu_n+s-\lambda}$, we have $\hat{B}_n(s) = \hat{b}_n(s-\lambda)$.

Therefore (21) is equivalent to

$$\boxed{\hat{h}_n(s) = \hat{h}_{n-1}(s)(1-\rho_n)\hat{B}_n(s) + p\rho_n\hat{B}_n(s)} \tag{22}$$

Moreover, from the two queues case, we deduce

$$\hat{g}_n(s) = p\rho_n\hat{B}_n(s) + \hat{g}_{n-1}(s)\,\frac{\hat{b}_n(s)(1-\rho_n)}{1-q\rho_n\hat{b}_n(s)}$$

$$\hat{g}_n(s) = p\rho_n\hat{B}_n(s) + \hat{g}_{n-1}(s)(1-\rho_n)\hat{B}_n(s) \tag{23}$$

In parts I, II, we showed that $\hat{g}_n(s) = \hat{h}_n(s)$ for n=1,2. The equations (22), (23) prove that $g_n(s) = h_n(s)$ for any n. That is to say that, whatever the number of tandem queues, the input distribution and the output distribution are identically distributed.

Moreover, we show in a second paper [27] that these distributions fails to be Poisson. Indeed we prove in this second paper that the distribution of interinput and interoutput intervals are identically distributed. Therefore, it is obvious from this result that the flow between two any queues is not a Poisson process.

We easily verify that, when n tends to infinity, the interinput (or interoutput) times are exponentially distributed with parameter λq^{-1}.

ON A PROPERTY OF JACKSON NETWORKS AND A DECOMPOSITION OF A GENERAL NETWORK

We have shown that the only streams which possesses the property to be a Poisson process in a feedback tandem queueing system, are the external arrival process and the departure process. Now let R be a general network and on this network, let $i \leftrightarrow j$ be the following relation of equivalence : from i it is possible to go to j and from j it is possible to go to i. The classes of equivalence of this relation are subnetworks such that when a customer leaves a subnetwork he never returns it. We name them irreducible subnetwork. The global network can now be decomposed in these irreducible subnetworks as shown in figure 4.

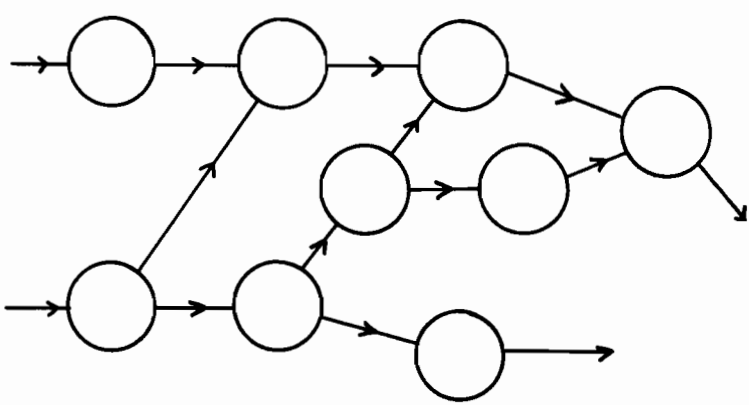
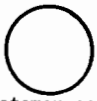


Figure 4



represents an irreducible sub network

When a customer goes out of an irreducible network he never returns.

For a given irreducible network we are going to examine the distribution of departure processes. From Kelly [25] we know that the customers leaving the system form a Poisson stream. Now from feedback property we have shown in the previous sections, that all the other flows are not Poisson processes. (Due to the irreducibility of the network, for each station we can build at least one tandem queueing system including this station, among the paths of the network).
This proves that all the flows except external arrival and departure streams are not Poisson.

If we come back to our global Jackson network we can conclude that the streams between two any stations of a same irreducible subnetwork are not Poisson but that external arrival and departure processes of each individual subnetwork are Poisson. The only Poisson processes in a Jackson network, are inter irreducible subnetwork streams. Indeed they are even the only renewal process as we show it in a further paper.

GENERAL QUEUEING NETWORK : A NEW APPROACH

We have shown that in a Jackson network only few streams have the property to be Poisson processes. Moreover, we have shown that input and output processes fail to be renewal, as soon as fedback customers are allowed. Now if we are interested by general queueing networks even streams without fedback customers are no longer renewal processes, but the dependances come only from the service time distributions (dependance by fedback customers are abolished). So, we propose the following new technique to study a general queueing networks :

1 - the network is decomposed into its irreducible sub systems;

2 - for each separate subnetwork, we compute the mean number of passage at each station;

3 - each station of each sub system is studied in isolation considering that the customers go through it one time and only one time.

We deal now with the last point in a more detailed manner; the decomposition has been studied in the previous section and the computation of the mean number of passage at each station for each sub system is obvious.

Study of each station. After to have decomposed our global network, we study each subsystem independently. For a given irreducible subnetwork, let λ be the arrival rate entering this subsystem and $e = (e_1,e_2,\ldots,e_n)$ be the solution of $e=q+eQ$ where $q=(q_{01},q_{02},\ldots,q_{0n})$ q_{0i} is the probability to enter this sub-system by station i : $Q = \{q_{ij}, 1 \leq i \leq n, 1 \leq j \leq n\}$ is the matrix of transition probabilities where n is the number of stations in the irreducible subnetwork. The quantity e_k, $k=1,..,n$ gives the mean number of passages in the station k. (*)

Our new approach consists to assume that the customers go through the station only one time (there is no longer fedback customers), see figure 5. So we are going to define an equivalent service time and a new input process. The new service time of a station is the total time spent by a customer in its server. For station k, $k=1,\ldots,n$, the new service time distribution (see figure 5) is defined by its rate $\tilde{\mu}_k=\mu_k/e_k$ and its squared coefficient of variation (SCV) $\tilde{Ks}_k=p_k+K_{s_k}(1-p_k)$ with $p_k = \max\left(\frac{e_k-1}{e_k},0\right)$ (if e_k is less than 1, namely the mean number of passage is less than 1, then $p_k=0$ and $\tilde{K_s}=K_s$).

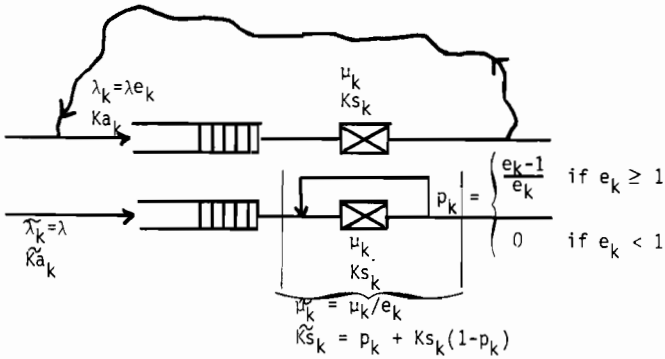


Figure 5

(*) Let λ' and e'_k be the total arrival rate in the general network and the mean number of passage computed on the general network. Generally e_k is different of e'_k, but $\lambda' e'_k=\lambda e_k$. e_k is the mean number of passage through station k knowing the customer enters the subnetwork we consider.

The equivalent arrival process is then defined assuming that all the customers entering the subnetwork go through each station one time and only one time. The equivalent arrival rate is $\lambda_k=\lambda$, k=1,...,n. Before to compute the SCV of the new arrival process we need to know the SCV of the initial arrival process. This one can be computed from several manners : Kobayashi [3], Gelenbe and Pujolle [5] or Sevcik et al [2]. We shall adopt a simplification of the second one (see Gelenbe and Mitrani [12]) :

$$Ka_k = \frac{1}{\lambda_k} \sum_{j=0}^{n} [(c_j-1)q_{jk}+1)]\lambda_j q_{jk} \qquad \text{with } \lambda_k = \lambda e_k$$

where c_j is the SCV of the inter departure times at station j and whose an approximated value is :

$$c_j = \frac{\rho_j^2(Ks_j-1)}{1-(1-\rho_j)q_{jj}^2} + 1, \qquad j=1,...,n.$$

c_0 is the SCV of the inter arrival times of external customers.

Now, the SCV of the arrival process at station k, k=1,...,n is $\tilde{Ka}_k=p_k+Ka_k(1-p_k)$.

Now, we have characterized each station by their two first moments of service and input processes, we have only to choose an adequate formula giving the response time and the mean number in station.

Firstly we have examined the case of a single server queue GI/G/1. A certain number of approximated formulae can be used. We have compared them one another using the mean number in station. They are the following ones :

1 - Kingman [13] $\bar{n} = \rho[1 + \frac{(Ka+Ks)}{2(1-\rho)}]$

2 - formula obtained by diffusion approximations

2a - [3] $\bar{n} = \frac{\rho}{1-\rho}$ with $\rho = \exp[\frac{2(1-\rho)}{\rho Ka+Ks}]$

2b - [14] $\bar{n} = \rho[1 + \frac{\rho Ka+Ks}{2(1-\rho)}]$

3 - Page [15] $\bar{n} = \rho+KaKs\bar{n}_{M/M/1}+Ka(1-Ks)\bar{n}_{M/D/1}+Ks(1-Ka)\bar{n}_{D/M/1}$
where $\bar{n}_{G/G/1}$ is the mean number in the station G/G/1 (for $\bar{n}_{D/M/1}$ we have used Kingman's fomula);

4 - Kramer and Langenback - Belz [16] :

$$\bar{n} = \rho[1 + \frac{\rho(Ka+Ks)}{2(1-\rho)} \times g]$$

$$\text{with} \quad g = \begin{cases} \exp[-\frac{2(1-\rho)(1-Ka)^2}{3\rho(Ka+Ks)}] & \text{if } Ka < 1 \\ \exp[-\frac{(1-\rho)(Ka-1)}{Ka+4Ks}] & \text{if } Ka \geq 1. \end{cases}$$

Other formulae have been examined as Marchal [17] and Kingman [18] ones, but they are bounds and not very accurate approximations.

Among the previous expressions 1, 3 and 4 have the advantage to give the exact Khintchine - Pollaczek formula for M/G/1 queueing system.

We shall say that a formula is correct if the result has an accuracy of the same order as a simulation (10 % at the maximum). Diffusion formulae are correct only on small intervals, Page is almost always correct when Ka and Ks are less than 1; Kingman when Ka and Ks are close to 1. Finally formula 4 is very satisfying : for practically all the cases examined ($0 \leq Ka \leq 3$ and $0 \leq Ks \leq 3$) the accuracy is very good. So we have adopted this expression for the computation of the mean number in station in our approach of queueing systems.

Now, if many server queues exist, our approach is the following : for the aim to determine only the departure process, the servers are replaced by just one server whose distribution function G(t) is chosen to satisfy $G(t) = F(st)$ where s is the number of servers and F(t) the distribution function for the service time of any server of the many server queue. The accuracy of this approximation about the departure process is evaluated by Arjas [19]. With this intermediate approximation we can now calculate the equivalent arrival and service processes.

About the mean number in station some formulae can be used. We have compared their results :

1 - Kingman [20] : $\bar{n} = s\rho + \frac{Ka+\rho^2 Ks}{2(1-\rho)}$

2 - Sakasegawa [21] : $\bar{n} = s\rho + \frac{(Ka+Ks)\rho^{\sqrt{2(s+1)}}}{2(1-\rho)}$

3 - Pujolle [22] (obtained by diffusion)

$$\bar{n} = \sum_{n=1}^{\infty} n\, p(n)$$

if $n \leq s$ $\quad p(n) = [1-p(o)]\, c\, \hat{\rho}(1)\hat{\rho}(2) \ldots \hat{\rho}(n)$

if $n \geq s$ $\quad p(n) = [1-p(o)]\, c\, \hat{\rho}(1)\hat{\rho}(2) \ldots \hat{\rho}(s-1)\hat{\rho}(s)^{n-s+1}$

$$p(o) = \left[1+\rho(1)+\rho(1)\rho(2) + \ldots + \frac{\rho(1)\ldots\rho(s-1)}{1-\rho(s)}\right]^{-1}$$

and $$c = \left[\sum_{i=1}^{\infty} p(i)\right]^{-1} = \left[\hat{\rho}(1)+\hat{\rho}(1)\hat{\rho}(2) + \ldots + \frac{\hat{\rho}(1)\ldots\hat{\rho}(s-1)}{1-\hat{\rho}(s)}\right]^{-1}$$

with $$\hat{\rho}(n) = \exp\left[\frac{2(\lambda-n\mu)}{\lambda Ka+n\mu Ks}\right], \quad \rho(n) = \frac{\lambda}{n\mu}.$$

We have also compared the results obtained by Stoyan [23] and Brumelle [24] which worked on bounds.

Results of these comparisons show the good accuracy of Sakasagawa's formula when $Ka \leq 1.2$ and $Ks \leq 1.2$, that we shall adopt in these limits. Finally formula 3 is taken in the other cases. It remains correct when Ka and Ks are not too large : Ka and Ks less than 2.

Accuracy of the method and conclusion. Firstly we have to note that for a Jackson network the exact result is found again by our new approach. Then, we have compared the solution of our new approach with simulation results on different queueing systems. Mainly, we have used the central server model (with the main queue representing the CPU and the feedback queues, the swapping devices) and tandem queueing systems. These two types of systems can be considered as extreme cases : the former is an irreducible subnetwork by itself.

On table 2 we have compared the results obtained by our new approach to previous

methods : KÜhn [6], and diffusion approximations : Reiser and Kobayashi [26], Gelenbe and Pujolle [4]. The model we have adopted is shown in figure 6 and the values of the parameters in table 1. The results of a simulation conducted by KÜhn are given in table 2, and also the relative differences referred to simulation results of KÜhn.

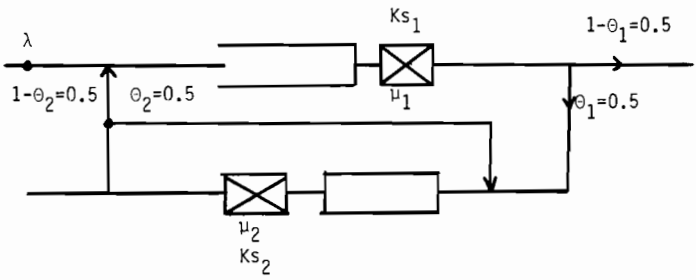


Figure 6

case	λ	μ_1^{-1}	Ks_1	μ_2^{-1}	Ks_2
1	0.5	0.9	0.5	0.84	0
2	0.5	0.95	0.5	0.89	0
3	0.5	0.9	1	0.84	1
4	0.5	0.95	1	0.89	1

Table 1

Case	Simulation Kühn 10^5 events		Theory Kühn		Theory Reiser Kobayashi		Theory Gelenbe Pujolle		Theory Pujolle Soula	
	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_1$	$\bar{n}_2$
1	7.36 0.45	4.16 0.14	6.12 16.8%	3.64 12.5%	5.76 8.1%	2.70 35%	6.65 9.6%	2.95 29%	7.53 2.3%	4.04 2.8%
2	13.69 1.32	5.84 0.35	12.47 8.9%	5.42 7.2%	14.30 4.4%	4.52 22.6%	12.62 7.8%	4.39 24.8%	14.61 7.1%	5.72 2%
3			9 exact	5.25	9.007	5.26	9.46	5.30	9 exact	5.25
4			19 exact	8.09	19.004	8.10	19.53	8.65	19 exact	8.09

With single server queues and SCV of service times less than 3, the results of all the tests we have made are in the confidence interval given by the simulation. With many server queues the results we obtained are of the same accuracy when the SCV of service times are between 1.5 and 0.5 and is not very far of the confidence interval when SCV of service times are less than 0.5 or between 1.5 and 2.

Finally, with our new approach we can predict with a good accuracy (better than with diffusion or iterative techniques) the behavior of a general single queueing system.

Another advantage is the possibility to study also many server queueing systems. In this last domain improvements are possible, especially by an accurate formula giving the mean number in the stations GI/G/s when SCV are either close to 0 or greater than 2.

ACKNOWLEDGEMENTS

We would like to thank Dr. D. Foley and Professor I. Mitrani for their comments and suggestions on the manuscript.

REFERENCES

[1] Chandy K.M., Herzog U., Woo L., "Parametric analysis of queueing networks" IBM J. Res. Develop. 19, 36-42, 1975.

[2] Sevcik K.C., Levy A.I., Tripathi S.K., Zahoyan J.L., "Improving approximations of aggregated queueing network subsystems", proc. of IFIP WG 7.3, Yorktown Heights, 1977.

[3] Kobayashi H., "Application of the diffusion approximation to queueing networks I", JACM 21, 2, 316-328, 1974.

[4] Gelenbe E., Pujolle G., "The behaviour of a single queue in a general queueing network", Acta Informatica 7, 123-136, 1976.

[5] Gelenbe E. Pujolle G., "A diffusion model for multiple class queueing networks", proc. of the 3rd International Symposium on Modelling and Performance Evaluation of Computer System, Bonn, 1977.

[6] Kühn P., "Analysis of complex queueing networks by decomposition", proc. of 8th International Teletraffic Congress, Melbourne, 1976.

[7] Labetoulle J, Pujolle G., "Modeling of packet switching communication network with finite buffer size at each node", proc. of IFIP WG 7.3, Yortown Heights, 1977.

[8] Burke P.J., "Proof of a conjective on the interarrival time distribution in a M/M/1 queue with feedback", IEEE Transaction on Communication 24, 175-178, 1976.

[9] Disney R., McNickle D.C., "The M/G/1 queue with instantaneous Bernoulli feedback", Technical report 77 - 4, University of Michigan, Ann Arbor, 1977.

[10] Takacs L., "A single server queue with feedback", Bell system Journal 42, 505-519, 1963.

[11] Melamed B., Zeigler B.P., Beutler F., "Simplifications of Jackson queueing network", technical report 163, University of Michigan, 1975.

[12] Gelenbe E., Mitrani I., "Analysis and synthesis of computer system models, to appear.

[13] Kingman J.F.C., "Some inequalities for the GI/G/1 queue", Biometrika 49, 315-324, 1962.

[14] Gelenbe E., "On approximate computer system models", J.ACM, 22, 2, 261-299, 1975.

[15] Page E., "Queueing theory in Operations research", Butterworths, 1972.

[16] Kramer W., Langenback-Belz M., "Approximate formulae for the delay in the queueing system GI/G/1", Proc. of 8th International Teletraffic Congress, Melbourne, 1976.

[17] Marchal W.G., "Some simple bounds and approximations in queueing", Research Report T. 294, George Washington University, 1974.

[18] Kingman J.F.C., "Inequalities in theory of queues", J.R. Statist. Soc. B. 32, 102-110, 1970.

[19] Arjas E., "Approximating many server queues by mean of single server queues", to appear in Mathematics of Operations Research (also Research Report D-21, University of Oulu, Helsinki, 1977).

[20] Kingman J.F.C., "The heavy traffic approximation in the theory of queues", Proc. Symposium on Congestion Theory, University of North Carolina, 137-169, 1964.

[21] Sakasegawa H., "An approximation formula $Lq \simeq \alpha\rho^{\beta}(1-\rho)$", Annals of the Institute of Statistical Mathematics 29, 1, 67-75, 1977.

[22] Pujolle G., "Files d'attente avec dépendance de l'éata" Proc. of SMF/AFCET Congrès, Palaiseau, 1978.

[23] Stoyan D., "Some bounds for many server systems GI/G/s", Math. Operations forsch. Statit. 5, 117-129, 1974.

[24] Brumelle S.L., "Some inequalities for parallel server queues", Oper. Res. 19, 402-413, 1971.

[25]Kelly F.P., "Networks of queues", Adv. Appl. Prob. 8, 416-432, 1976.

[26] Reiser M., Kobayashi H., "Accuracy of the diffusion approximation for some queueing systems", IBM J. Res. Devel. 18, 2, 110-124, 1974.

[27] Labétoulle J., Pujolle G., Soula C., "Distribution of the flows in a general Jackson network", to appear.

[28] Foley R.D., "On the output process of an M/M/1 queue with feedback", Research Report of the University of Michigan, 1977.

SPECIAL TOPICS

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

ON THE WORKING SET SIZE FOR THE MARKOV CHAIN MODEL OF PROGRAM BEHAVIOUR*

M. Hofri and P. Tzelnic
Dept. of Computer Science
Technion - Israel Institute of Technology
Haifa, Israel

ABSTRACT

The history of modelling of the address sequences generated by computer programs (often termed program behaviour) follows a familiar pattern: the better a hypothetical model fits experimental evidence, the less amenable it is for calculation. In this paper programs that generate successive page references which can be described by a first order general Markov chain are considered. We produce usable expressions for the distribution and moments of the steady state size of their Working Set of pages. These expressions are also specialized for the IRM and the Easton model. Only standard Markov Chain Theory is utilized.

Key Words: Program Behaviour; Paging Algorithms; Markov Chain Reference Model; Working set; Miss-rate function; Reverse Chain; Taboo Probabilities.

1. INTRODUCTION

It is well known that the performance of all virtual memory management policies depends significantly on the sequences of addresses generated by the active programs [1]. Two of the concepts that stand out in the attempts to understand the underlying processes are the Independent Reference Model (IRM) and the Working Set (WS) [1].

1.1 The first suggests a simple manner for the generation of page references by a program: Let $N = \{1,2,\ldots,n\}$ be the set of all pages of a program. The IRM stipulates that successive page references are independent, and that the page referenced at time t, $X(t)$, is i $(i \in N)$ with probability p_i; $\{p_i\}$ is a probability mass function (pmf) concentrated on N.

1.2 The second concept defines a set $W(t,\tau)$, comprising, at time t, all the distinct pages referenced through the course of the last τ references. It then suggests $W(t,\tau)$ as a good estimator of the 'locality of reference' at time t

* This paper is largely based on the work done by P. Tzelnic towards his Ph.D. degree.

and the 'near future'. Here τ - the WS 'window' - is a parameter of the model, and its choice determines the goodness of the estimation. We denote the size of $W(t,\tau)$ by $w(t,\tau)$. The WS policy is implemented by keeping in memory only the $w(t,\tau)$ pages of the Working Set.

The first moment of $w(t,\tau)$, the mean size of the working set, (or rather, its limit as $t\to\infty$, $w(\tau)$), was computed under various hypotheses on program behaviour. The simplest and best known of these results is the formula due to Denning and Schwartz [1], for the IRM.

1.3 In this paper we show how similar results and extensions can be obtained for a natural generalization of the IRM, i.e. the Markov Chain Model (MCM).

According to the MCM, $X(t)$ is a Markov chain satisfying certain mild regularity assumptions. This model is expected to describe 'real life' better than the IRM, even if for no other reason than the removal of the unnatural assumption of independence between successive page references (note that the IRM is a special case of the MCM). See [2] for an articulate justification of the MCM.

1.4 In Section 2 we present an analysis of the working set characteristics, in the frame of the MCM. We obtain the pmf of the WS size, and calculate its first two moments at steady state. As a direct consequence, the miss-rate function for a system employing the WS policy in its page replacement altorithm is evaluated.

In Section 3 we show that Denning and Schwartz's formula is indeed a special case of our results. We also use our results for Easton's special MCM [3].

2. WORKING SET DISTRIBUTION

2.1 Let the page reference generator be a first order, time-homogeneous, Markov chain over the set of states $N = \{1,\ldots,n\}$. Further, assume it is ergodic (in particular - irreducible and aperiodic). Let its transition matrix be P. We identify the state of the chain at time t, $X(t)$, with the page referenced at time t. Conditioned on the page reference generated at time t, the probability of the next - $(t+1)$st - page reference is then given by

$$\Pr[X(t+1) = j \mid X(t) = i] = P_{ij} \quad . \tag{2.1}$$

Let $\vec{\pi}$ be the steady state distribution of this chain ($\vec{\pi}P = \vec{\pi}$). Let $D(\vec{\pi})$ denote the diagonal matrix whose elements are π_i, $i = 1,\ldots,n$; and let $\tilde{P}$ denote the matrix transpose of P.

2.2 Following Kemeny and Snell [4], we define the process which is the reverse of $X(t)$ as that process which corresponds to the evolution in reverse order of the original chain. It is further assumed that the process $X(t)$ has as initial distribution (at $t = 0$) its stationary distribution. This last assumption makes ([4]) the reverse process thus defined a Markov chain, with the transition matrix:

$$\hat{P} = [D(\vec{\pi})]^{-1} P D(\vec{\pi}), \tag{2.2}$$

(or term wise, $\hat{P}_{ij} = \pi_j P_{ji}/\pi_i$).

Furthermore, the matrix $\hat{P}$ possesses the same steady state distribution, $\vec{\pi}$, as the direct chain.

From the matrix $\hat{P}$ we produce n matrices $\hat{P}_j$. Such a matrix is the zero matrix, with the exception of column j, which is column j of $\hat{P}$. We also find use for the n matrices $\hat{\underline{P}}_j$, defined by

$$\hat{\underline{P}}_j = \hat{P} - \hat{P}_j, \qquad j \in N, \tag{2.3}$$

i.e. - identical to $\hat{P}$ with the exception of a zero column j.

The reverse chain is a natural tool to use when the dynamics of the WS of the program is considered, since given $X(t)$, it is $\hat{P}$ that governs its characteristics directly.

2.3 In the following propositions we assume that the initial memory state (= the set of program pages that is in main memory) is of no import - or, alternatively, that t is "large enough".

The WS at time t, $W(t,\tau)$, is defined in terms of the process $\{X(t)\}$ as $W(t,\tau) = \{i | i \in \{X(t-\tau+1), X(t-\tau+2),\ldots,X(t-1), X(t)\}\}$, and $w(t,\tau)$ is the number of members of this set.

Proposition 1: The conditional probability that the Working Set size at time t is k, given that page i is referenced at time $t+1$, is:

$$\Pr[w(t,\tau)=k | X(t+1)=i] = \sum_{\ell=1}^{k} (-1)^{k-\ell} \binom{n-\ell}{k-\ell} \sum_{1\le j_1<\ldots<j_\ell\le n} \sum_{j=1}^{n} (\hat{P}_{j_1}+\ldots+\hat{P}_{j_\ell})^{\tau}_{ij} . \tag{2.4}$$

(See the note below on the perhaps peculiar conditioning.)

Proof. Sketchily presented, the argument runs along the following lines:

1. Given the state of the memory reference process at time $t+1$, τ transitions governed by $\hat{P}$ determine $W(t,\tau)$.

2. It is convenient to interpret the event $\{w(t,\tau) = k\}$ as "the program

avoids precisely $n-k$ pages during $t-\tau+1$ to t".

3. Denote by $A_{I|i}$ the event: $\{X(t-\tau+1),\ldots,X(t)\}$ avoid the set of pages I, given that $X(t+1) = i$.

4. Let $S_{r|i}$ denote the sum of probabilities $\sum_{I} \Pr[A_{I|i}]$ the summation being carried over all sets I s.t. $|I| = r$.

5. Then by 2. above and the inclusion - exclusion principle [5] follows

$$\Pr[w(t,\tau) = k|i] = \sum_{v=J}^{k} (-1)^v \binom{n-k+v}{v} S_{n-k+v|i} \cdot$$

6. The probability, that starting from i, the next (under $\hat{P}$) τ references are all within a set of ℓ pages J_ℓ, is given by:

$$\sum_{r=1}^{n} (\hat{P}_{j_1} +\ldots+ \hat{P}_{j_\ell})^{\tau}_{ir}, \qquad J_\ell = \{j_1,\ldots,j_\ell\} .$$

7. Using 4. and 6., we obtain for the value of S_r conditioned on $X(t+1) = i$:

$$S_{r|i} = \sum_{J_{n-r}} \sum_{j=1}^{n} (\hat{P}_{j_1} +\ldots+ \hat{P}_{j_{n-r}})^{\tau}_{ij}, \quad J_{n-r} = \{j_1 \cdots j_{n-r}\},$$

the summation being on all sets of size $n-r$.

8. Substituting the last result into 5., and putting $k-v = \ell$, yields Equation (2.4) (when we note that $S_n = 0$). □

Using Proposition 1, we obtain by removing the conditioning on $X(t+1)$:

Corollary 1: The steady state pmf of the working set size is:

$$\Pr[w(\tau) = k] = \sum_{\ell=1}^{k} (-1)^{k-\ell} \binom{n-\ell}{k-\ell} \sum_{(1\le j_1<\ldots<j_\ell\le n)} \sum_{i,j=1}^{n} \pi_i (\hat{P}_{j_1} +\ldots+ \hat{P}_{j_\ell})^{\tau}_{ij}. \qquad (2.5)$$

Note. It is perhaps useful to remark that Proposition 1 was stated using an "unnatural" conditioning - normally $W(t,\tau)$ is of interest at time t, when $X(t+1)$ is not known. Indeed, $\Pr[w(t,\tau) = k \mid X(t) = i]$ can be evaluated as well, and one readily obtains

$$\Pr[w(t,\tau)=k|X(t)=i] = \sum_{\ell=1}^{k} (-1)^{k-\ell} \binom{n-\ell}{k-\ell} \sum_{J_\ell} \left[\frac{k-\ell}{n-\ell}\right]^{I_{i\notin J_\ell}} \sum_{j=1}^{n} (\hat{P}_{j_1} +\ldots+ \hat{P}_{j_\ell})^{\tau}_{ij} ,$$

where I_A is 1 when A holds and 0 otherwise.
This however is a considerably more complex expression; specifically, a decomposition similar to (2.7) below cannot be affected on it. However, our main interest lies with the unconditional distribution $\Pr[w(t) = k]$; we obviously have

$$\Pr[w(t,\tau) = kj] = \sum_{i=1}^{n} \Pr[w(t,\tau) = k \mid X(t) = i]\Pr[X(t) = i]$$

$$= \sum_{j=1}^{n} \Pr[w(t,\tau) = k \mid X(t+1) = j]\Pr[X(t+1) = j],$$

and further, in the limit $t \to \infty$ both randomizations use the same pmf $\vec{\pi}$; hence Corollary 1 is indeed the result we need, the 'detour' noted here notwithstanding.

2.4 Using Eq. (2.5) the moments of the steady state WS size can now be computed: Denote the mean by $S(\tau)$:

$$S(\tau) = E[w(\tau)] = \sum_{k} k \Pr[w(\tau) = k].$$

Proposition 2.

$$S(\tau) = n - \sum_{k=1}^{n} \sum_{i,j=1}^{n} \pi_i (\hat{P}_k)^{\tau}_{ij} . \tag{2.6}$$

Proof:

$$S(\tau) = \sum_{k=1}^{n} k \Pr[w(\tau) = k] = \sum_{k=1}^{n} k \sum_{\ell=1}^{k} (-1)^{k-\ell} \binom{n-\ell}{k-\ell} S_\ell , \tag{2.7}$$

where

$$S_\ell = \sum_{i,j=1}^{n} \pi_i \sum_{1 \le j_1 < \ldots < j_\ell \le n} (\hat{P}_{j_1} + \ldots + \hat{P}_{j_\ell})^{\tau}_{ij} .$$

By changing the order of summation over k and ℓ in (2.7), and summing over $u = k-\ell$, from zero to n, we obtain:

$$S(\tau) = \sum_{\ell=1}^{n} S_\ell \sum_{u=0}^{n-\ell} (-1)^u \binom{n-\ell}{u} (u+\ell) .$$

The sum over u can be split into two components:

$$\ell \sum_{u=0}^{n-\ell} (-1)^u \binom{n-\ell}{u} = \ell \delta_{\ell,n} \quad ; \quad \sum_{u=0}^{n-\ell} (-1)^u u \binom{n-\ell}{u} = -\delta_{\ell,n-1} . \tag{2.8}$$

Therefore:

$$S(\tau) = nS_n - S_{n-1} =$$

$$= n \sum_{i=1}^{n} \pi_i \sum_{j=1}^{n} (\hat{P})^{\tau}_{ij} - \sum_{i,j=1}^{n} \pi_i \sum_{1 \le j_1 < \ldots < j_{n-1} \le n} (\hat{P}_{j_1} + \ldots + \hat{P}_{j_{n-1}})^{\tau}_{ij} =$$

$$= n - \sum_{i,j=1}^{n} \pi_i \sum_{k=1}^{n} (\hat{P}_k)^{\tau}_{ij} .$$

(See paragraph 2.8 concerning the evaluation of these quantities). □

In an entirely analogous manner we also obtain

$$S^{(2)}(\tau) = E[w^2(\tau)] = n^2 S_n - (2n-1)S_{n-1} + 2S_{n-2} = n^2 - (2n-1) \sum_{i,j=1}^{n} \pi_i \sum_{k=1}^{n} (\hat{P}_k)^{\tau}_{ij} +$$

$$+ 2 \sum_{i,j=1}^{n} \pi_i \sum_{1 \le j_1 < \ldots < j_{n-2} \le n} (\hat{P}_{j_1} + \ldots + \hat{P}_{j_{n-2}})^{\tau}_{ij} \qquad (2.9)$$

whence the variance is immediate.

2.5 A related interesting result is the expression for the miss-rate at equilibrium, $M(\tau)$, in the present model, under the WS policy. $M(\tau)$ is then defined as $\lim_{t\to\infty} \Pr[X(t) \notin W(t-1,\tau)]$. This is derived by observing that the present model satisfies the assumptions of [1], by virtue of which the following relation was shown to hold:

$$M(\tau) = S(\tau+1) - S(\tau) \; . \qquad (2.10)$$

Corollary 2:

$$M(\tau) = \sum_{k=1}^{n} \sum_{i,j=1}^{n} \pi_i [(\hat{P}_k)^{\tau}(I-\hat{P}_k)]_{ij} \; . \qquad (2.11)$$

2.6 We now present alternative expressions for $S(\tau)$ and $M(\tau)$. These perhaps are not as intuitive in derivation as the preceding ones, but are computationally superior.

Proposition 3:

$$S(\tau) = \sum_{s=0}^{\tau-1} \sum_{j,k=1}^{n} \pi_k (\hat{P}_k)^{s}_{kj} \quad , \qquad (2.12)$$

$$M(\tau) = \sum_{k,j=1}^{n} \pi_k (\hat{P}_k)^{\tau}_{kj} \quad . \qquad (2.13)$$

Proof: (Directly from Proposition 2):

$$S(\tau) = n - \sum_{k,j=1}^{n} U_{jk}(\tau) \qquad (2.14)$$

where $U_{jk}(\tau) = \sum_{i=1}^{n} \pi_i (\hat{P}_k)^{\tau}_{ij}$.

A recursive evaluation for these $U(\tau)$ follows:

$$U_{jk}(\tau) = \sum_{i=1}^{n} \pi_i \sum_{h=1}^{n} (\hat{P}_k)_{ih} (\hat{P}_k)_{hj}^{\tau-1}$$

$$= \sum_{h=1}^{n} \left\{ \sum_{i=1}^{n} \pi_i \hat{P}_{ih} - \sum_{i=1}^{n} \pi_i (\hat{P}_k)_{ih} \right\} (\hat{P}_k)_{hj}^{\tau-1}$$

$$= \sum_{h=1}^{n} \left\{ \pi_h - \sum_{i=1}^{n} \pi_i P_{ik} \delta_{kh} \right\} (\hat{P}_k)_{hj}^{\tau-1}$$

$$= U_{jk}(\tau-1) - \pi_k (\hat{P}_k)_{kj}^{\tau-1} \quad . \tag{2.15}$$

This difference equation for $U(\tau)$ should satisfy the initial condition:

$$U_{jk}(1) = \pi_j - \pi_k \delta_{jk} \ , \tag{2.16}$$

and thus yields

$$U_{jk}(\tau) = \pi_j - \pi_k \sum_{s=0}^{\tau-1} (\hat{P}_k)_{kj}^{s} \quad . \tag{2.17}$$

Substituting in (2.14) we obtain

$$S(\tau) = n - \sum_{k,j=1}^{n} \pi_j + \sum_{k,j=1}^{n} \pi_k \sum_{s=0}^{\tau-1} (\hat{P}_k)_{kj}^{s}$$

$$= \sum_{s=0}^{\tau-1} \sum_{j,k=1}^{n} \pi_k (\hat{P}_k)_{kj}^{s} \quad .$$

$$M(\tau) = S(\tau+1) - S(\tau) = \sum_{j,k=1}^{n} \pi_k (\hat{P}_k)_{kj}^{\tau} \ . \qquad \square$$

2.7 Remarks

1. The last relations suggest the following calculation scheme:
   (a) $S(1) = 1$,
   (b) $M(\tau)$ as given in (2.13),
   (c) $S(\tau+1) = S(\tau) + M(\tau)$.

2. Using a familiar representation of the mean WS size [1],

$$S(\tau) = \sum_{s=0}^{\tau-1} \left(1 - \sum_{k=1}^{n} \pi_k F_k(s)\right), \tag{2.18}$$

where $F_k(\cdot)$ is the distribution function for the interreference interval for page k, we get, by comparison with (2. 2):

$$F_k(s) = 1 - \sum_{j=1}^{n} (\hat{P}_{-k})^s_{kj} \ . \tag{2.19}$$

(This is actually a compact way to represent summation over all suitable realizations of sample paths.)

2.8 Complexity of Calculations It is of some interest to evaluate the number of operations involved in the various expressions obtained for $S(\tau)$, $M(\tau)$, etc. (essentially additions and multiplications of elements of $\hat{P}$ and $\vec{\pi}$ - no account is taken of loop control variables, the calculation of $\vec{\pi}$ itself and various 'bookkeeping' chores). Equation (2.4) requires, for each k and i, $n\tau \sum_{\ell=1}^{k} \ell\binom{n}{\ell}$, and for the complete (conditioned) pmf, for every i, $n^2\tau(n+1)2^{n-2}$ operations. Covering all i adds a factor of $2n$. A straightforward calculation of Equation (2.6) requires $n^2(n-1)^2\tau$ operations (an initial study indicates that this figure can be rather simply reduced by a factor of n at least, by using $\vec{\pi} = \vec{\pi}\hat{P}$. The calculation is more complex to control, however). Equation (2.12), using the procedure of 2.7.1, can be done in $n^3\tau$ operations (approximately the same as the elaborate calculation of (2.6)).

3. SPECIAL CASES

As an illustration let us evaluate the above expressions for two simple models of program behaviour.

3.1 Independent Reference Model Given that $P_{ij} = p_j$, $\sum_{j=1}^{n} p_j = 1$ the following results for the IRM follow:

$$\begin{aligned} \pi_i &= p_i \\ \hat{P}_{ij} &= P_{ij} = p_j \\ (\hat{P}_{-k})^\tau_{ij} &= (1-p_k)^{\tau-1}(1-\delta_{jk})p_j \end{aligned} \tag{3.1}$$

for $i,j = 1,\ldots,n$, $\tau \geq 1$.

Hence:

$$S(\tau) = n - \sum_{k=1}^{n} \sum_{i,j=1}^{n} \pi_i (\hat{P}_k)^{\tau}_{ij}$$

$$= n - \sum_{k} \sum_{ij} p_i p_j (1-p_k)^{\tau-1} (1-\delta_{jk})$$

$$= n - \sum_{k=1}^{n} (1-p_k)^{\tau} \quad , \tag{3.2}$$

consistently with the results in [1].

3.2 Easton's Model [3] This model makes use of a Markov chain of special structure. It is claimed to be a good description of interactive Data Base reference strings [3]. The transition probabilities are given by:

$$\begin{aligned} P_{ii} &= \alpha_i = 1-r + r\lambda_i \\ P_{ij} &= \beta_j = r\lambda_j, \; i \neq j \; , \end{aligned} \tag{3.3}$$

where $0 < r \leq 1$, $\sum_{i=1}^{n} \lambda_i = 1$ and $\lambda_i > 0$ for $i = 1,\ldots,n$.

For this model, we can easily verify that:

$$\begin{aligned} \pi_i &= \lambda_i \\ \hat{P}_{ij} &= P_{ij} \qquad i,j = 1,\ldots,n \quad . \end{aligned} \tag{3.4}$$

In order to compute $S(\tau)$ for Easton's model, we use the following relation, which is proved below:

$$\sum_{j} (\hat{P}_k)^{m}_{ij} = (1-\beta_k)^{m-1} [(1-\beta_k)(1-\delta_{ik}) + \delta_{ik}(1-\alpha_k)] \; . \tag{3.5}$$

With this it follows:

$$\sum_{i,j=1}^{n} \pi_i (\hat{P}_k)^{\tau}_{ij} = (1-\beta_k)^{\tau-1} \sum_{i} \lambda_i [(1-\beta_k)(1-\delta_{ik}) + \delta_{ik}(1-\alpha_k)]$$

$$= (1-\beta_k)^{\tau-1} \sum_{i} \lambda_i [(1-\beta_k) + \delta_{ik}(r-1)]$$

$$= (1-\beta_k)^{\tau-1} [1-\beta_k + \lambda_k(r-1)]$$

$$= (1-\beta_k)^{\tau-1}\,(1-\beta_k + r\lambda_k - \lambda_k)$$

$$= (1-\beta_k)^{\tau-1}\,(1-\lambda_k)$$

Hence:

$$S(\tau) = n - \sum_{k,i,j=1}^{n} \pi_i (\hat{\underline{P}}_k)^{\tau}_{ij} = n - \sum_{k=1}^{n} (1-\lambda_k)(1-r\lambda_k)^{\tau-1}\,, \qquad (3.6)$$

which is identical with Easton's result.

To prove the relation (3.5), one may use a probabilistic approach as follows: $\sum_j (\hat{\underline{P}}_k)^m_{ij}$ is actually the taboo probability of not entering state k, following an exit from state i, through m successive steps (along any sample path of the reverse Markov chain, $\hat{X}(t)$).

Two cases here have to be considered, corresponding to $i \neq k$ or $i = k$ in the initial state. Due to the simple structure of the Markov chain, the above probability is $(1-\beta_k)^m$, in the first case, and $(1-\beta_k)^{m-1}(1-\alpha_k)$, in the second one.

We also present a computational proof, which has the side benefit of yielding an explicit representation of the powers of $\hat{\underline{P}}_k$. This proceeds along the following lines:

1. We show inductively that:

$$(\hat{\underline{P}}_k)^m_{ij} = \begin{cases} \beta_j \dfrac{(1-\beta_k)^m - (1-r)^m}{r - \beta_k}\,, & j \neq k,\ j \neq i,\ i \neq k \\ \beta_i \dfrac{(1-\beta_k)^m - (1-r)^m}{r - \beta_k} + (1-r)^m, & j = i,\ i \neq k \\ \beta_j\,(1-\beta_k)^{m-1}\,, & j \neq k,\ i = k \\ 0\,, & j = k \end{cases} \qquad (3.7)$$

For $m = 1$, the above expressions are easily verified. Assuming they hold for m, we establish that they hold for $m + 1$ as well. (Note that the case $j = k$ is obvious.)

(a) $i \neq k,\ j \neq i,k.$

$$(\hat{P}_k)^{m+1}_{ij} = \sum_{h=1}^{n} (P_k)^m_{ih} (P_k)_{hj} = \sum_{h \neq k} (P_k)^m_{ih} P_{hj}$$

$$= (P_k)^m_{ii} P_{ij} + (P_k)^m_{ij} P_{jj} + \sum_{h \neq k,i,j} (P_k)^m_{ih} P_{hj}$$

$$= \left[\beta_i \frac{(1-\beta_k)^m - (1-r)^m}{r - \beta_k} + (1-r)^m \right] \beta_j + \beta_j \frac{(1-\beta_k)^m - (1-r)^m}{r - \beta_k} \alpha_j$$

$$+ \beta_j \frac{(1-\beta_k)^m - (1-r)^m}{r - \beta_k} \sum_{h \neq k,i,j} \beta_h$$

$$= \beta_j \frac{(1-\beta_k)^m - (1-r)^m}{r - \beta_k} \left[\alpha_j + \sum_{h \neq j} \beta_h \right] - \beta_j \left[\frac{(1-\beta_k)^m - (1-r)^m}{r - \beta_k} \beta_k - (1-r)^m \right]$$

$$= \beta_j \left[\frac{(1-\beta_k)^m - (1-r)^m}{r - \beta_k} (1-\beta_k) + (1-r)^m \right] = \beta_j \frac{(1-\beta_k)^{m+1} - (1-r)^{m+1}}{r - \beta_k} .$$

(b) $i \neq k,\ j = i.$

$$(\hat{P}_k)^{m+1}_{ii} = \sum_{h \neq k} (P_k)^m_{ih} P_{hi} = (P_k)^m_{ii} P_{ii} + \sum_{h \neq k,i} (P_k)^m_{ih} P_{hi}$$

$$= \left[\beta_i \frac{(1-\beta_k)^m - (1-r)^m}{r - \beta_k} + (1-r)^m \right] \alpha_i + \beta_i \frac{(1-\beta_k)^m - (1-r)^m}{r - \beta_k} \sum_{h \neq k,i} \beta_h$$

$$= \beta_i \frac{(1-\beta_k)^m - (1-r)^m}{r - \beta_k} \left[\alpha_i + \sum_{h \neq i} \beta_h \right] + (1-r)^m \alpha_i - \beta_i \beta_k \frac{(1-\beta_k)^m - (1-r)^m}{r - \beta_k}$$

$$= \beta_i \frac{(1-\beta_k)^m - (1-r)^m}{r - \beta_k} (1-\beta_k) + (1-r)^m (1-r+\beta_i)$$

$$= \beta_i \left[\frac{(1-\beta_k)^m - (1-r)^m}{r - \beta_k} (1-\beta_k) + (1-r)^m \right] + (1-r)^{m+1}$$

$$= \beta_i \frac{(1-\beta_k)^{m+} - (1-r)^{m+1}}{r - \beta_k} + (1-r)^{m+1}$$

(c) $i = k$, $j \neq k$.

$$(\hat{P}_{\sim k})^{m+1}_{kj} = \sum_{h\neq k} (P_{\sim k})^{m}_{kh} P_{hj} = \sum_{h\neq k,j} (P_{\sim k})^{m}_{kh} P_{hj} + (P_{\sim k})^{m}_{kj} P_{jj}$$

$$= \beta_j (1-\beta_k)^{m-1} \sum_{h\neq k,j} \beta_h + \beta_j (1-\beta_k)^{m-1} \alpha_j$$

$$= \beta_j (1-\beta_k)^{m-1} \alpha_j + \sum_{h\neq j} \beta_h - \beta_j (1-\beta_k)^{m-1} \beta_k$$

$$= \beta_j (1-\beta_k)^{m-1} (1-\beta_k) = \beta_j (1-\beta_k)^{m} \quad .$$

2. Summing over j in the various expressions in (3.7), we immediately obtain the stated result. □

Note. Both examples above happen to possess self-dual transition matrices (i.e., $\hat{P} = P$). This is neither a requirement of any of the procedures developed here, nor does it help in the calculation.

4. CONCLUSION

In this paper it was shown how quantities related to the WS of a program can be handled, when the program obeys the Markov Chain Model. Specifically, we produced expressions for the mean and variance of the WS size at equilibrium.

Although the calculations are inherently of the path counting type, closed expressions were obtained for two special popular models of program behaviour.

Further activity, using this approach, is aimed at deriving results for programs which display special structure (but not as degenerate as the above two cases).

5. ACKNOWLEDGEMENTS

The careful reading and copious comments of R. Syski of the University of Maryland helped remove several ambiguities and imprecisions. The note following Corollary 1 was motivated by a query of an attentive referee. We thank both.

REFERENCES

[1] Denning, P.J., Schwartz, S.C.: Properties of the Working Set Model, Comm. ACM 15, 3 (March 1972), pp. 191-198.

[2] Courtois, P.J., Vantilborgh, H.: A Decomposable Model of Program Paging Behaviour, Acta Inf. 6 (1976), pp. 251-275.

[3] Easton, M.C.: Model for Interactive Data Base Reference Strings, IBM J. Res. Develop. 19, 6 (Nov. 1975), pp. 550-557.

[4] Kemeny, J.G., Snell, J.L.: Finite Markov Chains, D. Van Nostrand, Princeton, 1960.

[5] Feller, W.: An Introduction to Probability Theory and its Applications. I, 3rd. Ed. 1968, J. Wiley, p. 106.

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

SELECTING PARAMETER VALUES FOR SERVERS OF THE PHASE TYPE

Edward D. Lazowska
Department of Computer Science
University of Washington
Seattle, Washington

and

Clifford A. Addison
Department of Computer Science
University of Toronto
Toronto, Ontario

One source of error in queueing network models of computer systems is the insufficiently accurate representation of service time distributions. When unacceptable error results from characterizing a service time distribution by its mean alone, the modeller will usually turn to a first-come-first-served service discipline and a phase-type server capturing additional characteristics of the observed service time distribution. Until recently, though, effective use of the method of phases has been inhibited by the absence of an efficient, soundly-based algorithm for parameter value selection.

In this paper, we approach the parameter selection problem by exploiting a recent result: in a wide class of queueing network models, performance measures are fully determined by the Laplace transforms of the various service time distributions evaluated at certain specific points. By selecting the parameter values of the observed service time distribution, rather than the moments of this distribution, increased accuracy in predicting performance measures can be achieved. We present a computationally efficient parameter selection procedure, and discuss experimental results.

INTRODUCTION

Considering their inherent limitations, simple queueing network models using exponential servers have demonstrated a remarkable ability to predict computer system performance accurately [Graham 1978]. It is an unfortunate fact of life, though, that significant errors sometimes arise in using them to model even seemingly straight-forward computer systems. An insufficiently accurate characterization of one or more service time distributions is often a major cause of this error.

The traditional approach to reducing this error involves representing the offending service center by a first-come-first-served service discipline and a phase-type server with parameter values selected so that its mean and variance equal the corresponding characteristics of the observed service time distribution. While queueing networks with one or more such service centers are not amenable to analysis by efficient exact solution techniques [Baskett et al. 1975], recent advances in approximate solution techniques, such as the Chandy-Herzog-Woo

theorem [Chandy et al. 1975], allow them to be analyzed in a computationally efficient manner.

In selecting parameter values for a phase-type server, a variety of characteristics of the observed service time distribution could be used. The decision to match its first two moments is an arbitrary one that stems from several practical considerations. First, the mean performance measures for M/G/1 queueing systems are fully determined by the first two moments of the service time distribution; the informal application of this result to queueing networks has great intuitive appeal. Second, until recently no algorithms existed to select parameter values based on more complex criteria.

The "two moment" approach to parameter value selection has a number of drawbacks, however. Consider the simple phase-type server illustrated in Figure 1, a two-stage hyperexponential. By appropriate assignment to parameters μ_1, μ_2, and p (where the μ's are service rates), this server can be made to match any mean and any coefficient of variation greater than one. (The coefficient of variation is equal to the standard deviation divided by the mean.) For this reason, the two-stage hyperexponential is commonly used to represent the CPU in queueing network models of computer systems. Because there are three parameters but only two constraints, selecting parameter values on this basis fails to exploit fully the flexibility of the server. In and of itself this would not be serious, but several authors (Price [1976], Lazowska [1977]) have recently noted that when used to represent the CPU in a queueing network model, various two-stage hyperexponential servers with identical means and coefficients of variation can result in dramatically different predictions for certain performance measures.

Table 1 is excerpted from [Lazowska 1977]. It shows the CPU utilization predicted by a queueing network model of the University of Toronto Computer Centre's IBM System/370-165 when various servers are used to represent the CPU. Included

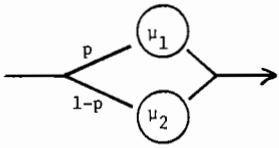


Figure 1

server	predicted utilization
exponential	83%
two-stage hyper.	
high extreme	79%
balanced load	69%
match skewness	64%
low extreme	56%

Table 1

are an exponential server matching the observed mean service time, and four two-stage hyperexponential servers, each of which matches both the observed mean and the observed coefficient of variation. Subject to these constraints, parameter values for the two-stage hyperexponentials were selected respectively to obtain the highest predicted CPU utilization, to balance the load on the two stages (the most common approach, in practice [Sauer & Chandy 1975]), to match the third moment of the observed service time distribution (note that this is not always possible with this server), and to obtain the lowest predicted CPU utilization. (The observed CPU utilization was 74%, which is within the range obtainable by the model.)

We reproduce this table in order to emphasize that selecting parameter values for a phase-type server based on the first two moments of the observed service time distribution is not necessarily adequate. And, as mentioned earlier, the classical general approach to selecting parameter values based upon additional characteristics of distributions, namely Prony's method [Cox 1955], suffers from a number of severe disadvantages [Bux & Herzog 1977a]. In response, several authors have recently devised new parameter selection methodologies.

In a series of papers ([Leroudier & Schroeder 1974], [Leroudier & Schroeder 1975], [Schroeder 1977]), Leroudier and Schroeder discuss a maximum likelihood approach to parameter selection, and demonstrate its use by matching the probability density functions of several distributions arising in an IBM CP/CMS system at the University of Grenoble. They do not describe modelling experiments that make use of the resulting servers.

More recently, Bux and Herzog [1977b] have described a technique that selects parameter values based upon the mean, the second moment, and an arbitrary number of points on the cumulative distribution function of an observed service time distribution. In essence, they apply a non-linear optimization procedure, increasing the number of stages in the server until a prescribed tolerance is reached. Again, they do not discuss the use of the resulting servers in queueing network models.

These two approaches to parameter selection represent a substantial advance, but still suffer from a potentially significant limitation: complexity. No guidance is given concerning the number of points of the observed distribution function that should be matched, the optimal location of those points, or the tolerance that should be prescribed. Matching a large number of points to a small tolerance may require a large number of stages in the server, a proportionately large investment of time in parameter value selection, and a proportionately large computational effort in analyzing the queueing network model in which the server is employed.

Here, we present a new approach to parameter value selection for servers of the phase type. Inspired by Bux and Herzog, we use a non-linear optimization

procedure. Instead of attempting to match a number of points of the observed cumulative distribution function, though, we match the value of the Laplace transform of the observed service time distribution at certain specific points. The impetus for this approach is provided by a recent result of Lazowska's [1977]: in a wide class of queueing network models, performance measures are fully determined by the Laplace transforms of the various service time distributions evaluated at certain specific points. In selecting parameter values according to this criterion, then, we are capturing exactly those properties of the observed distribution that are essential in the context of a queueing network model. This means that the flexibility of a specific server is exploited to the maximum possible extent. Further, the error in predicting performance measures will be proportional to the error in matching the Laplace transform values of interest.

In the section that follows, we briefly review the Laplace transform result mentioned above. In subsequent sections we describe our parameter selection procedure, mention some experimental results, and conclude with some implications and open questions.

LAPLACE TRANSFORMS AND PERFORMANCE MEASURES

It is natural to seek an explanation for the phenomenon illustrated by Table 1: two-stage hyperexponential servers with the same means and coefficients of variation can yield dramatically different performance predictions when used in queueing network models. The key lies in a derivation due to Jaiswal [1968], which is discussed by Price [1976].

Jaiswal derives an expression for server utilization in the M/G/1 queueing system with a fixed number of customers, N. This queueing system is of relevance here because it is equivalent to a closed queueing network with N customers in which the central server has a general service time distribution with FCFS scheduling, and the N I/O service centers have identical exponential service time distributions with no queueing delays. This equivalent network is illustrated in Figure 2.

Jaiswal shows that server utilization in this system is a decreasing function of the Laplace transform of the CPU service time distribution, evaluated at the points $n\lambda$, $0 \leq n \leq N-1$, where λ is the common service rate of the I/O service centers. The Laplace transform of a service time distribution is the integral of the product of its probability density function and the negative exponential function:

$$L^*[s] = \int_0^\infty e^{-sx} f(x)\, dx \;, \quad L^*[0] = 1$$

While Jaiswal's result is formally applicable to an extremely restricted class of queueing networks, the data in Table 1 hints at a much broader applicability. The four two-stage hyperexponential distributions considered there have consider-

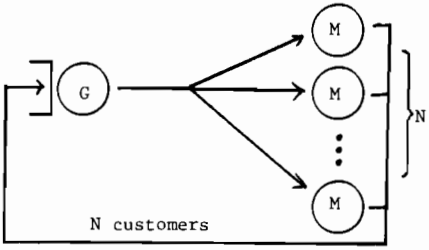


Figure 2

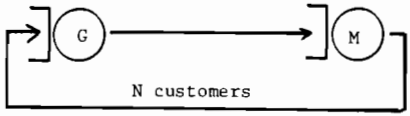


Figure 3

ably different probability density functions, and thus considerably different Laplace transform values, although their means and coefficients of variation are identical. But the queueing network model consists of a central server and a small number of dissimilar I/O service centers at which queueing may occur.

In extending the applicability of Jaiswal's result to other queueing network configurations, Lazowska [1977] considers the M/G/1 queueing system with finite waiting room. This queueing system is equivalent to a closed queueing network having N customers, a central server with a general service time distribution and FCFS scheduling, and a single, exponential I/O service center for which all customers must compete. This system, illustrated in Figure 3, represents the opposite end of the central server queueing network spectrum from that considered by Jaiswal and Price. Lazowska shows that server utilization is a decreasing function of the Laplace transform of the CPU service time distribution evaluated at the single point λ, the service rate of the I/O service center.

Based upon these two proofs and a number of examples, Lazowska argues the applicability of the result throughout the central server queueing network spectrum: CPU utilization, and thus all utilizations, are inversely proportional to the Laplace transform of the CPU service time distribution evaluated at points corresponding to the load dependent throughput rates of the I/O subsystem (equivalently, the load-dependent arrival rates of the CPU). Indeed, the intuitive explanation for this behavior, a "limited damage" argument involving residual lifetimes [Buzen & Shum 1977], applies to closed queueing networks of arbitrary topology.

The implication of these results for the problem considered in this paper is that the parameters of a phase-type server should be selected so that the Laplace transform of the server, evaluated at the load-dependent arrival rates of the service center, equal the corresponding values of the observed service time distribution.

MATCHING THE LAPLACE TRANSFORM

Our matching technique consists of two principal aspects: determining the necessary Laplace transform values of the observed service time distribution, and matching these values by varying the parameters of the phase-type server. As input data, we require a sequence of pairs $\{x_i, F(x_i)\}$ which specify the cumulative distribution function of the observed service time distribution. The x-values may be chosen for convenience in measurement. We also require an indication of the range of load-dependent arrival rates that must be considered. As output, we provide parameter values for a phase-type server. We shall make use of three numerical procedures: a cubic spline interpolation to provide specific cumulative distribution function values from the arbitrary ones furnished as input, a Gaussian integration to evaluate the transform of the observed service time distribution at appropriate values, and a non-linear least squares optimization to select parameter values for the phase-type server. We now describe the two apsects in more detail:

- Evaluating the Laplace transform of the observed distribution

We assume that our data has been obtained by monitoring the service center of interest over some period of time. Suppose that k customers passed through the service center during that period. Then we have the service times for each of these k customers, $x_1 \ldots x_k$. From this information, we must determine the value of the Laplace transform of the distribution at certain specific points.

Recall that the Laplace transform is defined by:

$$L^*[s] = \int_0^\infty e^{-sx} f(x)\, dx , \quad L^*[0] = 1$$

where $f(x)$ is a probability density function. Two difficulties arise in evaluating this integral. First, we must contend with the fact that the data from which we obtain our information concerning the underlying probability distribution is discrete. Moreover, obtaining approximations to $f(x)$ for specific x using data conveniently obtained from measurements may be somewhat tricky. Second, we must cope with an indefinite integral.

It is possible to rewrite the above expression in terms of $F(x)$, the observed cumulative distribution function, by taking advantage of the fact that if $L^*[s]$ is the Laplace transform of some function $G(x)$, then $sL^*[s]$ is the Laplace transform of its first derivative, $G'(x)$. We obtain:

$$L^*[s] = s \int_0^\infty e^{-sx} F(x)\, dx , \quad L^*[0] = 1 .$$

Working in terms of $F(x)$ has many advantages: the function is continuous and monotonically increasing, it ranges in value from zero to one, and from our data, we have a good idea of when $F(x)$ first becomes non-zero and when it first

becomes one. These properties enable us to obtain a good approximation to $F(x)$ for any value of x, using the following procedure:

- Find the largest, x_{max}, and the smallest, x_{min}, observed service times.
- Obtain a small number (15 is sufficient) of data points between x_{min} and x_{max}.
- Fit a piecewise cubic spline to these data points. This provides an interpolating function which is twice continuously differentiable, so that we have a reasonably smooth fit to the data.

Given any x, calculating the estimate to $F(x)$ involves finding the subinterval in which x lies, then evaluating the corresponding cubic polynomial.

Now that we have a procedure to approximate $F(x)$, we can concentrate on the integration itself. The first step is to eliminate the indefinite integral. We know that for $x \geq x_{max}$, $F(x) = 1$. Thus we have:

$$L^*[s] = s \int_0^{x_{max}} e^{-sx} F(x)\, dx + \int_{x_{max}}^{\infty} s e^{-sx}\, dx$$

$$= s \int_0^{x_{max}} e^{-sx} F(x)\, dx + e^{-sx_{max}}$$

Since we now have a definite integral and since the integrand is smooth, a general numerical integration procedure can be used to perform the integration. However, we can obtain a substantial savings in execution time if we tailor the procedure to the specific function. For example, the measurement and interpolation error in our values of $F(x)$ limits the accuracy of our values of $L^*[s]$, so we only require a modest degree of accuracy in the integration. We also know that e^{-sx} decreases rapidly as x increases. This means that beyond a certain point, $\hat{x}$, the integration need not be performed, since its contribution to the value of the integral is insignificant relative to the error already present.

To make the integration as efficient as possible, we break the range of integration into sub-intervals, with the interval size increasing as x increases. We use three-point Gaussian quadrature to evaluate the integral on each subinterval to within a specified tolerance, and this value is added to the overall sum. This approach allows us to halt when either the contribution of the subintervals becomes sufficiently small or x_{max} is reached.

We are now able to numerically evaluate the Laplace transform of an observed service time distribution for arbitrary values of the parameter s, given measured points on the cumulative distribution function. These points may be selected for convenience in measurement. It remains to select parameter values for the phase-type server to match the Laplace transform values of interest.

- Selecting parameter values for the phase-type server

  Here, we use the procedure described in the previous subsection to provide

specific values of $L^*[s]$. Given a phase-type server, we then choose parameter values so that its Laplace transform values match those of the observed distribution.

In this paper we shall deal with the three-stage, four-parameter server illustrated in Figure 4. (The μ's represent service rates, not times.) In a sense, this is more restrictive than the work of Bux and Herzog or Leroudier and Schroeder, who dynamically alter the number of stages used. Experience in modelling using this server, which we shall describe in a subsequent section, has led us to believe that it possesses the proper balance of simplicity and sufficiency. The general form of its probability density function is illustrated in Figure 5. By appropriate choice of parameter values, its characteristics can range from those of the hypoexponential to those of the hyperexponential. Additionally, it is not difficult to extend our method to more general server structures.

The Laplace transform of the server illustrated in Figure 4 is:

$$C^*[s] = \frac{\mu_1}{s + \mu_1} \left\{ \frac{p\,\mu_2}{s + \mu_2} + \frac{(1-p)\mu_3}{s + \mu_3} \right\}$$

Given a set of pairs $\{s_i, L^*[s_i]\}$ we wish to find values for μ_1, μ_2, μ_3, and p, such that the difference between $L^*[s]$ and $C^*[s]$ is minimized over the s_i. As noted, the crucial points are the load-dependent arrival rates of the service center in question. We choose, however, to match a large number (roughly 50) of points spread uniformly between the smallest and largest arrival rates. We do this to minimize the effect of the error inherent in the values of $L^*[s]$. This error is essentially random, with a mean of zero. By working with a large number of points, we can obtain results that are minimally influenced by this error. Finally, we constrain the parameter values so that the mean of the

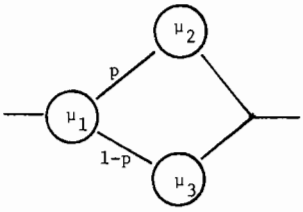


Figure 4

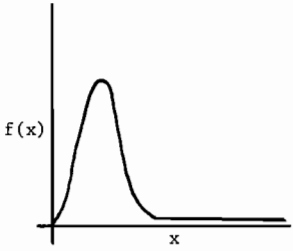


Figure 5

observed service time distribution is matched. (This will be discussed in a subsequent section.) We do this by introducing artificial data points at $s = 0.0001$, with a value selected such that the slope of the transform between zero and 0.0001 is equal to the measured mean.

The actual fitting is done by a non-linear least squares procedure from the Harwell subroutine package [Fletcher 1971].

The following is a summary of the matching procedure, which has been incorporated into a package with a simple interface:

- Obtain values of $F(x)$ at several data points.
- Interpolate these data points using piecewise cubic splines.
- Evaluate $L^*[s]$, using three-point Gaussian integration on sub-intervals, on a set of points spanning the range of interest. Introduce artificial data points to constrain the mean of the phase-type server.
- Perform the least-squares fit.

The running time of the algorithm, and the goodness of fit obtained, both increase with the number of points s_i that are employed. In our experiments with this methodology, some of which are reported in the next section, running times of under ten seconds on a 1 MIP machine have been experienced.

EXPERIMENTAL RESULTS

The parameter selection technique just described has been used successfully in several modelling studies. We describe a few of our experiences here.

● Characterizing service time distributions

Figure 6 roughly illustrates the probability density function for the CPU service time distribution of the University of Toronto Computer Centre system. In the model of this system discussed earlier, the load-dependent arrival rates to this service center were in the range between 0.2 and 0.9. Table 2 displays certain characteristics of the service time distribution.

The parameter selection technique just described results in a server whose Laplace transform values are within 1% of those shown in Table 2. The mean of the server is 8.7, and its coefficient of variation is 3.0. When this server is used in the model, predicted CPU utilization is 74%, precisely the observed value. In other words, the errors manifested in Table 1 are due entirely to the insufficiently accurate characterization of the CPU service time distribution, and are eliminated by our parameter selection technique.

In this same model, we have used our technique to select parameter values for a two-stage hyperexponential server. Because of the restricted flexibility of this server, its Laplace transform values are in error by as much as 10%. When used in the model, predicted CPU utilization is 75%, an error of under 2%.

As a matter of interest, Table 3 displays Laplace transform values for the

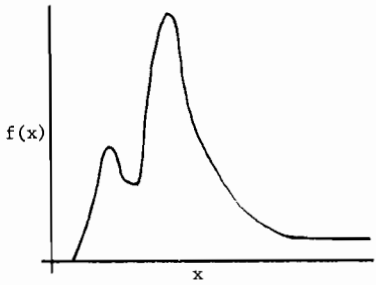


Figure 6

	observed distribution
mean	8.7
C.V.	12.7
L*[.0]	1.000
L*[.2]	.463
L*[.4]	.276
L*[.6]	.184
L*[.8]	.130
L*[1.]	.096

Table 2

two-stage hyperexponential servers at the low utilization and high utilization extremes in Table 1. It is interesting to note the wide discrepancy in Laplace transform values, since the servers are of the same form and have identical means and coefficients of variation.

● Characterizing response time distributions

Although the motivation for our parameter selection technique arose from the importance of the Laplace transform in characterizing service time distributions in queueing network models, the technique has been applied successfully in more general contexts.

Lazowska and Sevcik [1978] have shown that in a large class of queueing network models, the distribution of response times can be closely approximated by considering only "high-level" aspects of system behavior: the mean response time, and the distribution of the number of cycles through the network required by a customer. "Low-level" aspects of system behavior (service time distributions, scheduling disciplines, the presence of multiple customer classes, network topology and transition probabilities) influence the distribution of response

	low extreme	high extreme
mean	8.7	8.7
C.V.	12.7	12.7
C*[.0]	1.000	1.000
C*[.2]	.900	.454
C*[.4]	.824	.294
C*[.6]	.761	.217
C*[.8]	.706	.172
C*[1.]	.659	.143

Table 3

times only to the extent that they influence the mean response time of the customer class of interest. Their approximation technique proceeds as follows:

- Measure the system to obtain the parameters required by a queueing network model. In addition, obtain information about the distribution of the number of cycles through the system required by the customer class of interest. (Note that the queueing network model requires only the mean number of cycles as a parameter.)

- Analyze the queueing network model to obtain the mean response time of the customer class of interest. Divide this value by the mean number of cycles required by that customer class to obtain the mean duration of a single cycle.

- In parallel with the previous step, approximate the observed distribution of the number of cycles through the system using a phase-type server. Of course, the observed distribution will be discrete rather than continuous, but the difference is unimportant here.

- Each stage of the phase-type server obtained in the previous step represents some mean number of cycles through the system. At each stage, substitute the mean response time of a customer requiring that mean number of cycles. The result is a phase-type server representing the distribution of response times for the customer class of interest.

At the heart of this approximation technique is the phase-type characterization of the distribution of the number of cycles through the network required by a customer. It is here that the technique described in this paper is employed.

As a case study, Lazowska and Sevcik consider IBM's Time Sharing Option as implemented at the University of Toronto Computer Centre. Measurement data, including the distribution of the number of passages through the OS/MVT dispatcher per interaction,was gathered. Since a customer passes through the dispatcher after each I/O operation, this constitutes a natural definition of a cycle for this system. Figure 7 illustrates the observed probability density function of the number of cycles required per interaction. Using the methodology described

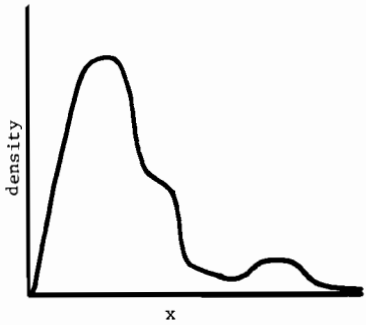


Figure 7

in this paper to select parameter values for the phase-type server illustrated in Figure 4 yields the following results:

p: 0.225 μ_1: 1/(3.16 cycles) μ_2: 1/(213 cycles) μ_3: 1/(2.58 cycles)

To obtain an approximation to the distribution of response times for this system, the mean response time (2.33 seconds) is divided by the mean number of cycles per interaction (53 cycles) to obtain the mean duration of a single cycle (0.044 seconds). (In practice, the mean response time would be provided by a queueing network model. For the purposes of this validation, however, it was obtained from measurement data.) Next, the substitutions described above are performed, resulting in the following phase-type approximation to the distribution of response times:

p: 0.225 μ_1: 1/(0.139 seconds) μ_2: 1/(9.370 seconds) μ_3: 1/(0.113 seconds)

Lazowska and Sevcik report good agreement between this approximation and the observed distribution of response times.

CONCLUSIONS

We have presented a new technique to select parameter values for phase-type servers based on the characteristics of observed distributions. In basing our algorithm on the Laplace transform evaluated at certain specific points, we have exploited a recent result in queueing network theory. As a consequence, we obtain good accuracy in performance prediction with a minimal investment in server complexity and modelling difficulty. Two specific applications are described; we believe that the technique will prove useful in many modelling contexts.

A number of questions are raised by the Laplace transform result itself. We are also investigating several issues more closely related to the subject of the present paper. In particular, we are attempting to study more closely the relationship between error in matching the Laplace transform and error in performance prediction. This work leads to the more general topic of error analysis in hierarchical queueing network models, which we are also investigating. There seem to be interesting relationships to the Chandy-Herzog-Woo theorem and to decomposability results in general.

ACKNOWLEDGEMENTS

We should like to thank Kenneth C. Sevcik of the Univeristy of Toronto's Computer Systems Research Group and James J. Horning of the Xerox Palo Alto Research Center for their generous assistance. This material is based upon work supported in part by the National Research Council of Canada, and in part by the National Science Foundation under Grant No. MCS78-04347.

REFERENCES

[Baskett et al. 1975]
Forest Baskett, K. Mani Chandy, Richard R. Muntz and Fernando G. Palacios; "Open, Closed and Mixed Networks of Queues with Different Classes of Customers"; JACM 22,2, April 1975.

[Bux & Herzog 1977a]
W. Bux and U. Herzog; "Approximation von Verteilungsfunktionen, ein wichtiger Schritt bei der Modellbildung fuer Rechensysteme"; Workshop ueber Modelle fuer Rechensysteme, March 1977.

[Bux & Herzog 1977b]
Werner Bux and Ulrich Herzog; "The Phase Concept: Approximation of Measured Data and Performance Analysis"; in K. M. Chandy and M. Reiser, eds., Computer Performance, North-Holland, 1977.

[Buzen & Shum 1977]
Jeffrey P. Buzen and Annie W. Shum, personal communication, February 1977.

[Chandy et al. 1975]
K. M. Chandy, U. Herzog and L. Woo; "Parametric Analysis of Queueing Networks"; IBM J. Res. Develop. 19,1, January 1975.

[Cox 1955]
D. R. Cox; "The Use of Complex Probabilities in the Theory of Stochastic Processes"; Proc. Cambridge Philosophical Society 51, 1955.

[Fletcher 1971]
R. Fletcher; "A Modified Marquardt Subroutine for Non-Linear Least Squares"; AERE R.6799, Harwell, 1971.

[Graham 1978]
G. S. Graham, editor; "Queueing Network Models of Computer System Performance"; Computing Surveys 10,3 (Special Issue), September 1978.

[Jaiswal 1968]
N. K. Jaiswal; Priority Queues; Academic Press, 1968.

[Lazowska 1977]
E. D. Lazowska; "The Use of Percentiles in Modelling CPU Service Time Distributions"; in K. M. Chandy and M. Reiser, eds., Computer Performance; North-Holland, 1977.

[Lazowska & Sevcik 1978]
Edward D. Lazowska and Kenneth C. Sevcik; "Exploiting decomposability to approximate quantiles of response times in queueing networks"; 2nd International Conference on Operating Systems, IRIA, October 1978.

[Leroudier & Schroeder 1974]
J. Leroudier and A. Schroeder; "Estimation of service times distribution for operating systems modelling"; Report 93, IRIA, December 1974.

[Leroudier & Schroeder 1975]
Jacques Leroudier and Anne Schroeder; "A Statistical Approach to the Estimation of Service Times Distributions for Operating Systems Modelling"; in E. Gelenbe and D. Potier, eds., International Computing Symposium 1975; North-Holland, 1975.

[Price 1976]
T. G. Price; "A Note on the Effect of the Central Processor Service Time Distribution on Processor Utilization in Multiprogrammed Computer Systems"; JACM 23,2, April 1976.

[Sauer & Chandy 1975]
C. H. Sauer and K. M. Chandy; "Approximate Analysis of Central Server Models"; IBM J. Res. Develop. 19,3, May 1975.

[Schroeder 1977]
Anne Schroeder; "Estimating Input Densities for Operating System Models"; in J. R. Barra et al. eds., Recent Developments in Statistics; North-Holland, 1977.

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

INTERLEAVED MEMORY SYSTEMS WITH MARKOVIAN REQUESTS

T.L. Török

Central Research Institute for Physics
Budapest, 1525, POB 49
Hungary

Abstract: A storage system model consisting of N identical modules is studied. The stream of subsequent requests forms a homogeneous Markov chain of finite state space, i.e. the probability of a certain request occurring depends solely on the type of the previous one. During a memory cycle each module can serve a single corresponding element of the reference list the order of which is invariable. The list is therefore disjoined into stages containing different elements. The length of these stages/memory bandwidth/ is investigated in equilibrium. Apart from the distribution and expectation of the length, approximations are also obtained. Certain specific transition probability matrices are given special consideration.

Introduction

One of the most important factor of program behaviour is an algorithm handling the memory. To choose an optimal algorithm means on large scale to choose an adequate model. For general purposes the interleaved memory seems to be an effective mean.

Several authors have investigated computer models with interleaved memory [1], [2], [3], [4], [6]/. Chang considered the two key questions as

- Dependence within the access sequence
- Queueing mechanism for accesses.

Present paper attempts to generalize previous results with respect to the dependence. [1], [3], [6].

I. The Model

Consider an interleaved memory system with N identical modules. The reference string / the sequence of requests for the modules / is assumed to be infinite. During a cycle time each module can serve a single request thus the number of busy

modules during a cycle / memory bandwidth denoted by B / characterizes the speed of the system. We have two extreme cases when the reference string consists of identical elements /B = 1/ and of 1,2,..,N,1,..,N,1,2,../B = N/, respectively. This illustrates that B depends on the form of the reference string. In general, the requests are stochastically generated. The independent uniform case was investigated by [2], [3], [6]. As a generalization let the reference string form a homogeneous Markov chain ξ_n with state space {1,2,..,N} and transition probability matrix P./i.e. the probability that a request for module i is followed by a request for module j is P_{ij};i,j = 1,...,N/. In this case, the memory bandwidth becomes a period of the sample function of ξ_n without repeating any states.

The investigation of Markov chains mostly comes to an end after having determined the stationary / equilibrium / distribution. This describes the asymptotic behaviour in concrete, individual moments independent of each other but fails to answer the questions of subsequent instants. It is obvious that

$$\lim P\{\xi_n = j,\ \xi_{n+1} = k\} \neq p_j \cdot p_k \quad . \tag{1}$$

In the following, an attempt is made to investigate a more global characteristic in equilibrium.

II. The Basic Problem

Let ξ_n be a Markov chain of finite state space {1,2,...,N} with the transition probability matrix $P = [p_{ij}]$. Let us define the sequence of random variables

$$\tau_o = 0 \quad , \quad \tau_n \equiv \min \{k > \tau_{n-1} \mid \xi_k = \xi_\ell;\ \tau_{n-1} \leq \ell < k\} \tag{2}$$

that is, the event $\{\tau_{n+1} = k\}$ means that all the variables ξ_{τ_n}, $\xi_{\tau_{n+1}}$,..., $\xi_{\tau_{n+k-1}}$ take different values but $\xi_k = \xi_\ell$ for some $\ell, \tau_n \leq \ell < k$. Thus the sample functions of ξ_n are divided into periods without repetition $/\nu_n \equiv \tau_{n+1} - \tau_n /$. The length of these periods will be investigated in the following:

Examples: 1. Recurrent events

$$P = \begin{bmatrix} q_o & p_o & 0 & 0 & 0 & \cdots \\ q_1 & 0 & p_1 & 0 & 0 & \cdots \\ q_2 & 0 & 0 & p_2 & 0 & \cdots \\ \vdots & & & & & \end{bmatrix}$$

Zero is the single state capable of being repeated. Assuming $\xi_o = 0$ the variables ν_n are independent, identically distributed and

$$P\{\nu_1 = k+1\} = P\{\xi_{k+1} = 0,\ \xi_k = k,\ldots,\xi_1 = 1 \mid \xi_o = 0\} = q_k \cdot \prod_{i=o}^{k-1} p_i \tag{3}$$

2. Random walk

$$p_{i,i-1} = q\ ,\ p_{i,i+1} = p\ ,\ p + q = 1\ ;\ i = 0,\pm1,\pm2,\ \ldots$$

in this case only the periods

$$i,\ i+1\ ,\ i+2\ ,\ \ldots\ ,\ i+k-1,\ i+k \quad (,i+k-1) \text{ and}$$

$$i,\ i-1\ ,\ i-2\ ,\ \ldots\ ,\ i-k+1,\ i-k \quad (,i-k+1)$$

are of length k+1. The corrresponding probabilities are $p \cdot q^k$ and $q \cdot p^k$, respectively. The variables ν_n are independent, identically distributed and

$$P\{\nu_1 = 1\} = 0$$

$$P\{\nu_1 = k\} = p^{k-1} \cdot q + q^{k-1} \cdot p \quad \text{if } k > 1 \tag{4}$$

In general, the variables ν_n fail to behave as simply as the above examples. Introducing the sequence $\eta_n \equiv \xi_{\tau_n}$ we have

Theorem 1. The sequences (η_n, ν_n) and η_n both form homogeneous Markov chains and the probabilities

$$P\{\nu_n = k \mid \eta_{n-1} = i\}$$

depend only on i.

Proof.

$$P\{\eta_n = i_n, \nu_n = k_n \mid \eta_r = i_r, \nu_r = k_r;\ r < n\} = (1-\delta_{i_n i_{n-1}}) \times \sum_{\ell=u_{n-1}} \sum_{s_j}^{\cdot)}$$

$$P\{\xi_{u_n} = i_n,\ \xi_\ell = i_n,\ \xi_r = s_r;\ u_{n-1} \le r < u_n,\ r \ne \ell \mid \xi_{u_{n-1}} = i_{n-1},\ A^{n-1}\} +$$

$$\delta_{i_n} \cdot i_{n-1} \sum_{s_j}^{*} P\{\xi u_n = i_n,\ \xi_r = s_r,\ u_{n-1} \le r < u_n \mid \xi_{u_{n-1}} = i_{n-1},\ A^{n-1}\} \quad ,$$

where $u_n = \sum_{i=o}^{n} k_j$, $A^{n-1} = \bigcup_{m=o}^{n-1} A_m$ and A_m is the Borel field corresponding to $\xi_m \cdot \Sigma^{\cdot)}$ and Σ^* mean k_n-2 and k_n-1 sequential summing, respectively for all

$$s_{u_{n-1}+1} \ , \ s_{u_{n-1}+2} \ , \ \cdots \ , \ s_{u_n-1}$$

with the restriction $s_i \ne s_j$ if $j < i$, and neither $s_i = i_{n-1}$ nor $s_i = i_n$, except the prescribed case $s_\ell = i_n$ in the first term.

The homogeneous Markov property of ξ_n calls forth the idea that the above conditional probabilities do not depend on A^{n-1} nor on n:

$$Q_{ij}(k) = P\{\eta_n = j \ , \ \nu_n = k \mid \eta_{n-1} = i\} = (1-\delta_{ij}) \times$$

$$\times \sum_{\ell=1}^{k-1} \sum_{s_1 \ne i,j} p_{is_1} \sum_{s_2 \ne i,j,s_1} p_{s_1 s_2} \cdots \sum_{s_{\ell-1} \ne \cdots} p_{s_{\ell-2} s_{\ell-1}} p_{s_{\ell-1} j} \cdots \sum_{s_{k-1} \ne \cdots} p_{s_{k-1} j}^{-+}$$

$$+ \delta_{ij} \sum_{s_1 \ne j} p_{js_1} \sum_{s_2 \ne j,s_1} p_{s_1 s_2} \cdots \sum_{s_{k-1} \ne j,s_r} p_{s_{k-1} j} \quad . \tag{6}$$

Note. 1. (η_n, ν_n), or more exactly (η_n, τ_n), is a Markov renewal process of discrete time. We have the renewal time μ_i of type i:

$$P\{\mu_i = k\} = P\{\nu_n = k \mid \eta_{n-1} = i\} \tag{7}$$

In the examples, we obtained simple renewal processes on the one hand because $\eta_n \equiv 0$, and on the other $P\{\mu_i = k\}$ does not depend on i.

2. (η_n,τ_n) is a special Markov reneval process, namely the corresponding semi-Markov process ξ_n is a Markov chain. This is reflected when generating the transition probabilities /6/.

The theory of Markov renewal processes [5] establishes the equilibrium length of the periods ν_n. Let

$$Q_{ij} = \sum_{k=1}^{\infty} Q_{ij}(k) \text{ and } h_i(k) = \sum_j Q_{ij}(k) = P\{\mu_i = k\}$$

If $\underline{f}^T = (f_1,\ldots,f_N)$ is the fixed point of the stochastic matrix $Q = [Q_{ij}]$ and $\sum_{i=1}^{N} f_i = 1$ / that is, $\underline{f}^T$ is the stationary distribution of the imbedded Markov chain η_n / then

$$P\{B = k\} = \lim_{n\to\infty} P\{\nu_n = k\} =$$

$$= \lim_{n\to\infty} \sum_{i=1}^{N} P\{\mu_i = k \mid \eta_{n-1} = i\} \cdot P\{\eta_{n-1} = i\} =$$

$$= \lim_{n\to\infty} \sum_{i=1}^{N} h_i(k) \cdot P\{\eta_{n-1} = i\} = \sum_{i=1}^{N} f_i \cdot h_i(k) \qquad (8)$$

III. Some Computing Difficulties, Approximations and Special Cases

The calculation of the stationary distribution $\underline{f}$ from the matrix Q takes $O(N^3)$ elementary operations and needs $O(N^2)$ memory places. It is much more difficult to determine the probabilities $Q_{ij}(k)$. For instance if N=2 then

$$Q(1) = \begin{bmatrix} p_{11} & 0 \\ 0 & p_{22} \end{bmatrix}, \qquad Q(2) = \begin{bmatrix} p_{12}p_{21} & p_{12}p_{22} \\ p_{21}p_{11} & p_{2i}p_{12} \end{bmatrix}$$

If $N \geq 3$ then we have two possibilities: either to store $O(N^{N+1})$ data or to produce all of the possible trajectories of periods. Their number is

$$S(N) = \sum_{\ell=1}^{N} \binom{N}{\ell} \cdot \ell \cdot \ell! = N! \cdot \sum_{\ell=1}^{N} \frac{\ell}{(N-\ell)!} > 2 \cdot N \cdot N!$$

Since $\frac{S(N+1)}{S(N)} > N+1$ the computing time increases very quickly.

If $p_{ij} = \frac{1}{N}$ then

$$Q_{ij}(k) = \frac{(k-1)\ (N-2)\ (N-3)\ \ldots\ (N-k+1)}{N^k} \qquad i \neq j$$

$$Q_{ii}(k) = \frac{(N-1)\ (N-2)\ \ldots\ (N-k+1)}{N^k}, \qquad (9)$$

because the numerator shows the number of terms in /6/ and N^k is the value of a single term. If $p_{ij} \neq \frac{1}{N}$ then /9/ can be regarded as an approximation which is not

too bad for large k.

An upper bound for E(B) is to be yielded in the following way. Let $P_o = I$, $P_1 = \phi(P)$ where $\phi(A) = A - \text{diag } A$ and let us form the sequence

$$P_n = \phi(\text{MIN}\{P_{n-1}P_1, P_1P_{n-1}\}) \tag{10}$$

where MIN is meant element-wise †. The i-th element $p_i^{(n)}$ of the row vector $P_n\underline{1}$ is an upper bound for $P\{\mu_i > n\}$ where $\underline{1}$ is a column vector all components of which are equal to 1. Thus

$$E(B) \leq \max_{1\leq i\leq N} \sum_{k=o}^{N} p_i^{(k)} \quad . \tag{11}$$

This bound can only be improved at the considerable expense of losing sight of its "side" /upper or lower/. The number of terms contained by $P\{\mu_i > k\}$ is $\prod_{i=1}^{k} (N-i)$ and this number in $p_i^{(k)}$ is $(N-1)(N-2)^{k-1}$. Taking this into consideration

$$\sum_{k=o}^{N} \frac{\prod_{i=3}^{k}(N-i)}{(N-2)^{k-2}} \cdot p_i^{(k)} \tag{12}$$

can be a closer approximation than /11/ but, as indicated above, its side cannot be determined in general.

Our larger problem emerges when determining matrix Q since small changes in the matrix can cause large deviations in the fixed point $\underline{f}$. There exist special cases when $\underline{f}$ can be given without difficulty.

1. Cyclic random walk.

P has the following form:

$$p_{ij} = c_k \qquad \text{if } j-i \equiv k \bmod N \quad .$$

In this case one can easily verify that $\underline{\pi}^T = \underline{f}^T$ and $E(B) \leq \sum_{k=o}^{N} p_i^{(k)}$.

In general, if P is double stochastic or a Toeplitz matrix then Q fails to inherit the corresponding feature.

2. P has the form $p_{ij} = o$ if $o<j\leq i$ in which case the single repeating state is zero, i.e. $\underline{f}^T = (1,0,\ldots,0)$ and $B \equiv \mu_o$.

3. From the Kolmogorov cycle conditions and from /6/ it easily follows that if ξ_n is a reversible Markov chain* then so is η_n and $\underline{f} = \underline{\pi}$.

In general, there seems to be no explixit relation between the stationary distributions $\underline{f}$ and $\underline{\pi}$. Let

$$P\{B(\pi) = k\} = \sum_{i=1}^{N} \pi_i \, P\{\mu_i = k\}$$

---

†That is: $C = \text{MIN}(A,B)$ if $c_{ij} = (a_{ij}, b_{ij})$, $i,j = 1,2,\ldots,N$.

*ξ_n is reversible if $\pi_i p_{ij} = \pi_j p_{ji}$ where P_{ji} and π_i are the transition and stationary probabilities, respectively.

If N = 2 it is easy to verify that

$$P\{B>1\} \geq P\{B(\pi)>1\}$$

and thus

$$E(B) \geq E(B(\pi)) \quad .$$

However, for $N \geq 3$ it is extremely tedious to prove something similar but so far no example has been produced to refute the validity of /14/.

Finally it is worthwhile to mention that no numerical example seems to contradict the fact that

$$P\{B>k\} \leq \sum_{i=1}^{N} \pi_i p_i^{(n)} \quad .$$

IV. Applications and further Generalizations

It is not surprising that the distribution of the memory bandwidth /B/ is independent of the state starting from:

$$P\{B = k\} = \Sigma\, f_i \cdot h_i(k)$$

The value of E(B) plays a fundamental part in most of the known models which are involved in Case 1 of the previous section. They are of simple structure because the sequence of the requests forms a cyclic random walk with

$$P = \begin{bmatrix} \frac{1}{N} & \frac{1}{N} & \cdots & \frac{1}{N} \\ \frac{1}{N} & \frac{1}{N} & \cdots & \frac{1}{N} \\ \cdot & & & \\ \cdot & & & \\ \cdot & & & \\ \frac{1}{N} & \frac{1}{N} & \cdots & \frac{1}{N} \end{bmatrix} , \qquad P = \begin{bmatrix} \alpha & \beta & \alpha & \cdots & \alpha \\ \alpha & \alpha & \beta & \cdots & \alpha \\ \cdot & & & & \\ \cdot & & & & \\ \cdot & & & & \\ \alpha & \alpha & \alpha & \cdots & \beta \\ \beta & \alpha & \alpha & \cdots & \alpha \end{bmatrix}$$

respectively.

To generalize the queueing mechanism of the requests meets some difficulties. If we use a buffer of size Q for requests whose modules are busy then the study becomes extraordinarily difficult. /We have to deal with cycles containing, at most, Q identical elements./ For approximation purposes Hellerman's model is investigated admitting a buffer of size Q. It is not too difficult to ascertain that

$$P\{B(N,Q) = k\} = \frac{k \cdot k! \cdot S(Q+k,k)}{N^{k+Q+1}} \cdot \binom{N}{k} \tag{18}$$

where B(N,Q) is the memory bandwidth with N modules and buffer of Q, and S(n,k) are the Stirling numbers of the second kind/[7]/. If Q = 0 then we obtain Hellerman's formula /[3]/

$$P\{B = k\} = \frac{k \cdot k!}{N^{k+1}} \cdot \binom{N}{k} \tag{19}$$

Figure 1 shows E(B) versus N for different values of Q.

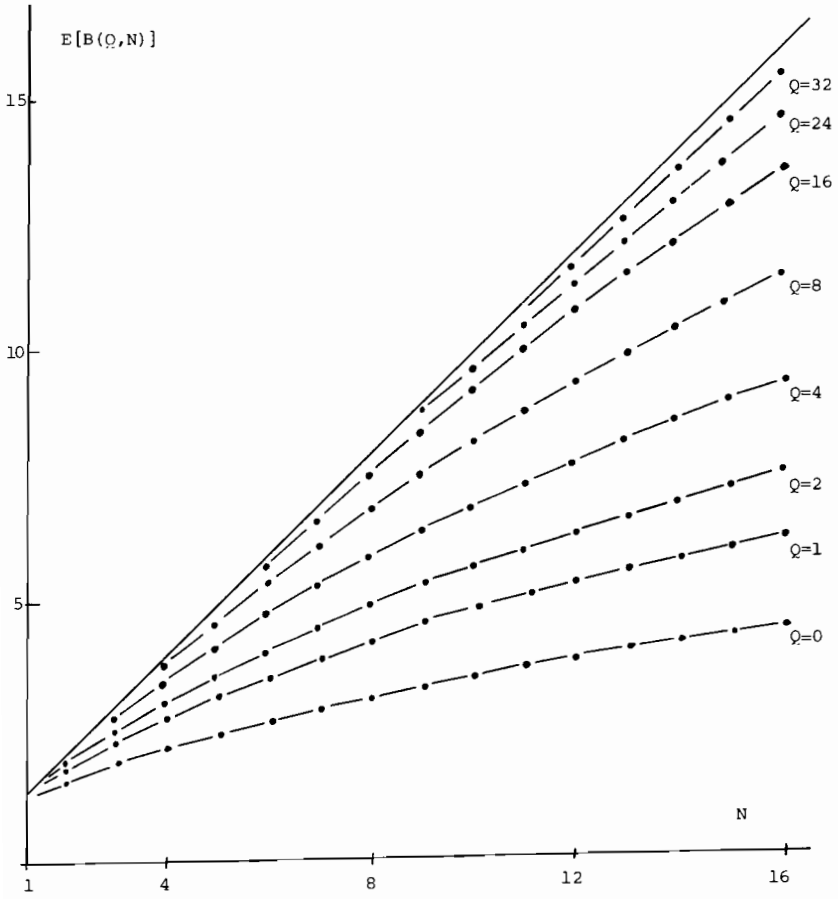


Figure 1

References

[1] G.J. Burnett and E.G. Coffman: Combinatorial Problem Related to Interleaved Memory Systems, Journal of ACM, 20/1973/, pp. 39-45.
[2] D.Y. Chang, D.J. Kuck and D.H. Lawrie: On the Effective Bandwidth of Parallel Memories, IEEE C-27 /1977/, pp. 480.
[3] H. Hellerman: Digital Computer Principles, McGraw Hill, New York, 1972.
[4] A. Iványi and I. Kátai: On the Performance of Computers with Interleaved Memory, Second Hungarian Computer Science Conference, Budapest, 1977.
[5] R. Pyke: Markov Renewal Processes, Ann. Math. Stat. 32, /1961/, pp. 1243-1259.
[6] C.V. Ravi: On the Bandwidth and Interference in Interleaved Memory Systems, IEEE C-22, 1972, pp. 899-901.
[7] J. Riordan: An Introduction to Combinatorical Analysis, Wiley, New York, 1958.

PERFORMANCE CONTROL

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

PERFORMANCE EVALUATION OF A CACHE MEMORY FOR A MINI-COMPUTER*

Marc BADEL, Jacques LEROUDIER
IRIA/LABORIA
BP 105
78150 LE CHESNAY
(FRANCE)

In this paper, we study the performance of a cache memory for a mini computer, namely a MITRA 125 manufactured by SEMS, France. The performance is evaluated in terms of hit ratios and speeding-ups of the target machine, via a simulator fed with address traces picked up during executions of real programs. Since such a performance strongly depends upon the behaviour of the programs analyzed, the impact of program behaviour on performance is carefully studied.

1 - INTRODUCTION

A cache memory [1, 2] is a buffer between the CPU of a computer and its main memory aiming at increasing the speed of the memory accesses. Usually in a computer, the CPU cycle is noticeably smaller than the read/write memory cycle, thus CPU is slowed down by memory. In this context, a cache represents a trade-off permitting to get better performance without using a fast but expensive technology for the entire memory of a computer [2, 3].

In our case, we study the gain, in terms of speeding-ups of the target machine, which can be expected by providing a mini-computer, namely a MITRA 125 [4], with a cache memory. FormerlyBELL and CASENT [5] have evaluated the improvement brought to a PDP/8 by a cache the structure of which was very close to ours. But they mainly emphasized the implementation, ignoring the influence of programs' behaviour on performance. This paper thoroughly analyzes the performance obtained with a cache and attempts to relate it to the behaviour of the programs running on the machine.

In a first section, we describe the mechanism of the cache under study for our purposes and we also define the performance criteria to be considered. In the second section we describe the tool used to pick up measurements and we analyze the characteristics of the collected measurements, that is mainly the programs'behaviour. The third section is devoted to performance analysis via an address trace driven simulator. This performance is evaluated both in a steady and a non-steady state context.

2 - THE CACHE

2.1 - The mechanism

The structure of the cache we study is presented in Figure 1. This structure does

* This work has been supported by SEMS (Société Européenne de Mini-Informatique et de Systèmes) under the contract IRIA/SEMS 02/78.

not make any assumption on the addressing mechanism of the target machine. In particular, the use of base registers by the MITRA 125 [4] is completely transparent. The accesses to the cache are made through the absolute address of the information in main memory.

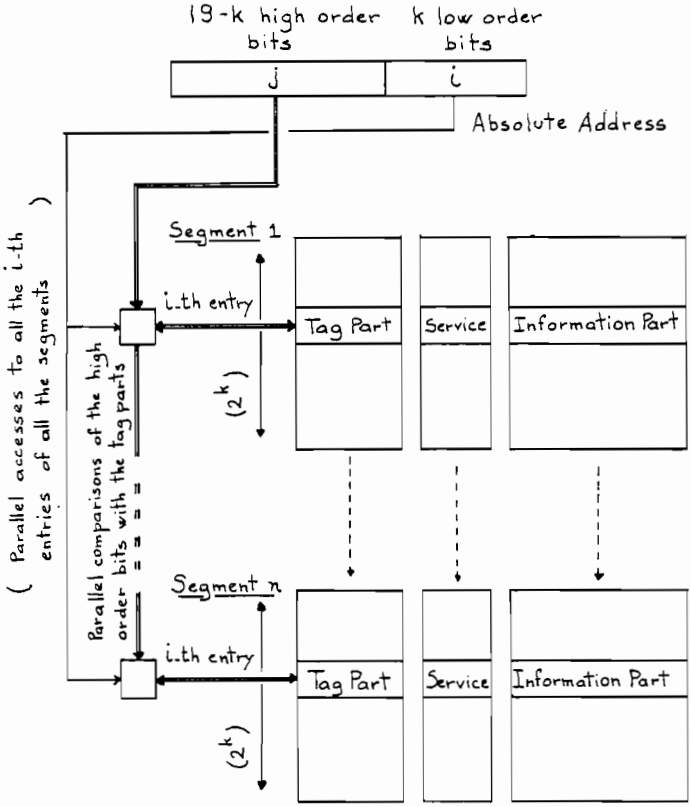


Figure 1 : Cache architecture

The absolute address is a 19-bit number used to reference a 16-bit word in memory. This address is divided into two parts : the right-most part is composed of the k low order bits of the address and the left-most part of the (19-k) other bits (high order bits). For clarity, let us assume that the address to be interpreted has a right-most part which is the number i and a left-most part which is the number j (see figure 1).

The cache is divided into n segments (n being a power of 2) of 2^k entries each, if k is the number of bits of the address low-order part. Thus the total cache size, as far as the information part is concerned, is of $2^k \times n$ 16-bit words ($2^k \times n$ double words with a prefetching mechanism). The n segments are accessed in parallel through the k low-order bits of the address. More precisely, all the i-th entries of each segment are accessed in parallel (see figure 1). Each cache entry consists of three parts (see figure 1) :

1 - a tag part of (19-k) bits which contains the high order bits of the address,

2 - a service part which contains validity and parity bits and also bits used by the replacement algorithm,

3 - an information part of 16 or 32 bits which contains the information itself (the content of the referenced word or double word according to the existence or not of prefetching).

The n i-th entries of each segment are scanned in order to find the tag equal to the number j (see figure 1). If such an entry exists, the referenced word is within the cache and the information part of the entry is transferred to CPU. If such an entry does not exist, the referenced work is not within the cache and it must be transferred to it from memory. If an i-th entry of a segment is still free, the referenced word is stored in it, otherwise the replacement algorithm has to free one of the n i-th entries.

It is noteworthy that the proposed architecture aims at exploiting a certain program "locality" [6] : the words having addresses the high-order parts of which are equal and the low-order parts successive (memory vicinity) will be loaded together into the cache.

2.2 - The performance criteria

To evaluate the improvement brought by the cache, we need to define some performance criterion. The level we consider is not detailed enough to take into account problems such as data or addresses interleaving, data path width, pipelinning, and so on, all related to parallel accesses. We shall consider some "speeding-up" of the machine.

Let us consider the mean time for a word (or a double word) to be available for CPU. Let m and M be such mean times according to whether the word is within the cache or not. Let p be the probability of finding a referenced word within the cache (p is also called the hit ratio of the cache). Then the mean time for a word to be available is :

$$T = pm + (1-p)M \tag{1}$$

Thus the CPU throughputs with and without cache, in terms of data words available to be processed by CPU, are respectively :

$$D_C = \frac{1}{pm + (1-p)M} \tag{2}$$

$$D_M = \frac{1}{M} \tag{3}$$

Therefore we can measure some increase of the speed of the machine by the ratio :

$$A = \frac{D_C}{D_M} = \frac{M}{pm + (1-p)M} = \frac{1}{1-p(1 - \frac{m}{M})} \tag{4}$$

It must be noticed that the improvement we measure is not the total resultant speeding-up of the machine since we do not consider parameters as possible parallel accesses, mean time per instruction processing and average number of accesses per instruction. It must be clear that the throughputs we consider are not instruction throughputs, but we think that equation (4) permits to roughly evaluate an order of magnitude of the speeding-up visible by a user.

It is evident from the equation (4) that in order to maximize A, we have to minimize the expression : $1-p(1 - \frac{m}{M})$, that is :

- p has to be as large as possible

- $\frac{m}{M}$ as small as possible.

The ratio $\frac{m}{M}$ is an architecture parameter which expresses the usefulness of the cache (m and M must be noticeably different). In our case, it is about $\frac{1}{4}$ ($\frac{m}{M} \simeq \frac{1}{4}$) and we shall consider it as fixed. The hit ratio p depends on the cache structure and the programs'behaviour. From the equation (4) we give some "speeding-ups" we can expect for different values of p.

p	0.0	0.5	0.6	0.7	0.8	0.85	0.9	0.95	1.0
A	1.	1.60	1.82	2.10	2.50	2.76	3.08	3.48	4.0

Thus, roughly speaking, we can see that to double the speed of the machine we must assure a hit ratio of 0.7 and to triple the speed a hit ratio of 0.9.

Therefore we shall study, in this paper, a structure of cache aiming at maximizing the hit ratio p.

3 - THE COLLECTED MEASUREMENTS

3.1 - The measurement characteristics

The optimization of the hit ratio p cannot be undertaken independently of the programs'behaviour, which is, in our case, the way the programs address words in main memory.

For that purpose, hardware probes were plugged into a MITRA 125 CPU and connected to another SEMS mini-computer (a SOLAR)*). Periodically the SOLAR copies on a magnetic tape the measurements it has picked up. Because of the way the measurements are collected, there exist "holes" on the tape which correspond to the measurements missed during copying (from 100 to 500 milliseconds). Thus measurements on the tape are gathered in blocks of 32K words of 32 bits separated from each other by "holes".

An elementary measurements (a referenced address) is coded on 32 bits and provides information on :

- the address referenced (19 bits)
- the nature of the access : data or instruction
- the base registers used
- the direction of the transfer between the CPU and the memory (write or read operation)
- the possible conflicts with i/o channels.

About thirty programs were run on the MITRA 125 and spied. We report, in this paper, the results concerning eight of them which are described in the following table :

*) all this part of work has been done by the SEMS Center at Echirolles (France)

Identification	Description
Program A	Fortran Program
Program B	Test of the MMT2 Monitor
Program C	MAS2 Assembler
Program D	Benchmark (coded in FORTRAN)
Program E	MAS Assembler with standard environment
Program F	First phase of MAS Assembler
Program G	Sorting program
Program H	Grammar Generator

We studied the behaviour of these different programs according to :

- the uses of the different base registers,
- the respective access rates to instructions or data,
- the respective read or write access rates,
- the conflict rates with i/o transfers,
- the behaviour within memory, that is with respect to the referenced addresses,

Since the program behaviour within memory is much more complex than the behaviour related to other parameters such as base register uses or different access rates, we shall emphasize this point by devoting it a complete section in the sequel of the paper.

We give in figure 2 some characteristics of program behaviour. Some of them are rather stable and repeated through all the programs :

- the read access rates are high and stable (always betwenn 83 % and 90 %). To explain this fact, notice that all the instruction accesses are read accesses.
- the instruction access rates are not very much stable, since they vary from 41 % to 68 % and the confidence interval widths can reach 4 %. But, however, they indicate a relative balance (around 50 %) between instructions and data.
- the conflicts with i/o transfers are either non-existent or very rare, except possibly for the programs E and F where they can reach 2 % and 3 % of all the references.

On the contrary, some other figures of merit heavily vary from one program to another :

- the base register G is relatively used, from a minimum of 13 % with program B (test of monitor) to a maximum around 70 % with programs F and H.

- the base register S (used by the only monitor) is very fluctuating. Its use is minimum for programs A (Fortran program) and D (Fortran benchmark), and maximum for programs C (assembler) and G (Sorting program). Notice that the use of base register S permits to roughly evaluate the overhead associated with each program.

- the base register Q (the base register of the shared sub-programs) is seldom used except by programs A, B and D. This base register seems to be associated with FORTRAN.

- the base register Z (the base register of the shared data) is not very used. This should change in the future, in the case of interactive systems (conversational systems, data bases, and so on).

- the base register H permits to evaluate the frequence of instructions moving bytes strings in memory.

Programs	Read Access Rates in %	Conflict Rates with I/O transfers in %	Instruction Access Rates in %	Access Rates using base G in %	Access Rates using base Z in %	Access Rates using baseQ in %	Access Rates using base S in %	Access Rates using baseH in %
A	83.00 ∓ 0.08	no conflict	42.6 ∓ 0.2	45.5 ∓ 0.4	0.05 ∓ 0.05	29.6 ∓ 0.7	0.3 ∓ 0.2	24.5 ∓ 0.5
B	89.6 ∓ 0.7	1.6 ∓ 0.5	68. ∓ 2.	13. ∓ 1.	4.3 ∓ 0.3	45. ∓ 3.	26. ∓ 2.	8.6 ∓ 0.5
C	88.0 ∓ 0.2	0.8 ∓ 0.2	55.1 ∓ 0.2	44. ∓ 1.	3.7 ∓ 0.1	0.33 ∓ 0.09	48.4 ∓ 0.9	3.1 ∓ 0.3
D	87.8 ∓ 0.2	0.3 ∓ 0.2	58.1 ∓ 0.3	30.3 ∓ 0.2	0.5 ∓ 0.1	62. ∓ 1.	4. ∓ 1.	2.9 ∓ 0.2
E	85.2 ∓ 0.5	2. ∓ 0.5	53.3 ∓ 0.5	57. ∓ 3.	2.7 ∓ 0.2	1.1 ∓ 0.3	29. ∓ 2.	8.3 ∓ 0.5
F	85.3 ∓ 0.6	1.1 ∓ 0.6	51.0 ∓ 0.5	73. ∓ 2.	1.8 ∓ 0.2	2.2 ∓ 0.9	14. ∓ 2.	7.3 ∓ 0.6
G	77.6 ∓ 0.8	2.9 ∓ 0.9	41.8 ∓ 0.8	20.4 ∓ 0.9	4.7 ∓ 0.3	0.8 ∓ 0.3	47. ∓ 1.	24. ∓ 1.
H	86.9 ∓ 0.8	0.8 ∓ 0.8	46.2 ∓ 0.5	71. ∓ 4.	1.6 ∓ 0.6	1.1 ∓ 0.6	19. ∓ 3.	6. ∓ 1.

Figure 2

It must be noticed that the program G significantly differs from the other ones, mainly because of numerous references issued by the i/o channels. In particular, this phenomenon affects the read access and instruction access rates.

We way wonder about the validity of our results, for the number of references analyzed varies from 325,445 to 1,295,419, which represents the execution of 100,000 to 600,000 instructions. This number can seem very low, but one should remember the technique which was used to collect the measurements. The referenced addresses were sampled per blocks of 32,768 consecutive addresses, each block being separated from the next one by a "hole" in time of about 100 to 500 milliseconds. Therefore the analysis we performed concerns relatively long program executions (about 10 minutes) and we can guess its validity.

These results lead to some first conclusions :

- the structure of the cache must not depend on the uses of the different base registers, for these uses vary too much from one program to another and, possibly, in the future.

- on the contrary, the structure of the cache may perhaps depend on the distribution between instructions and data by dividing the cache into two parts according to the measured distribution (roughly 50 % for each part).

3.2 - Program behaviour with respect of memory

In a first step, we visualized the behaviour of the different programs in memory. The figures in Annexe I display raw information that we dispose on each program.

The X-axes represent the execution time given in numbers of references and the numbers of the references pages (blocks of consecutive words in memory) are plotted on the Y-axes. A dash on the graph indicates the pages referenced by the program at a given instant of time.

Notice that, due to the scales used to plot the figures, several addresses seem to be referenced simultaneously. This is only an artefact. The scales which have been used are as follows :

- X-axes : the time unit is of 1024 references
- Y-axes : the memory address unit is of 1000 words.

Because of the technique used to collect measurements, magnetic tapes contains blocks of 32K consecutives references. We have visualized this fact in Annexe I by drawing on each figure vertical lines to separate the different blocks. It is noteworthy that, on each tape (thus for each program), blocks look very similar. This fact confirms that the data can be considered as relatively representative.

From these results we can deduce that :

- program A is "quiet" and stationary, except possibly on the block 16 where we notice the influence of printer i/o operations.

- program B is more "excited", perhaps representative of a heavy and varied workload. This program will be preferably considered to evaluate the cache performance in order to avoid performance overestimation.

- "stripes" of continuously referenced addresses can be noticed in the different programs.

- program H shows the influence of
  - addresses referenced by i/o transfers (block 17)
  - wait loops when the machine is idle (blocks 18, 19, 20, 21, 22).

In order to quantify this memory behaviour we studied some measure of program "locality" [6].

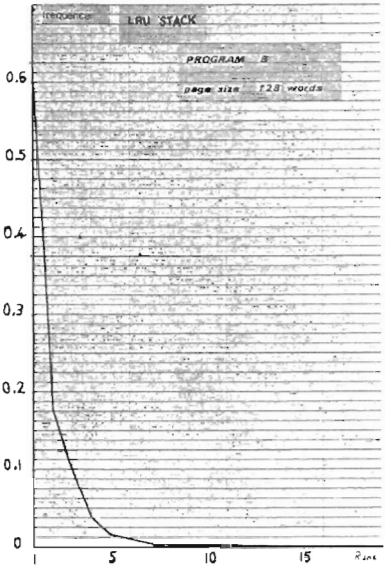
3.2.1 - LRU Stacks [7]

If some "locality" phenomenon exists the probability for an address to be referenced roughly decreases as its rank in the LRU stack increases. Therefore we simulated the LRU (Least Recently Used) algorithm [7] with the address references recorded on the magnetic tapes previously described. We have considered two kinds of stacks :

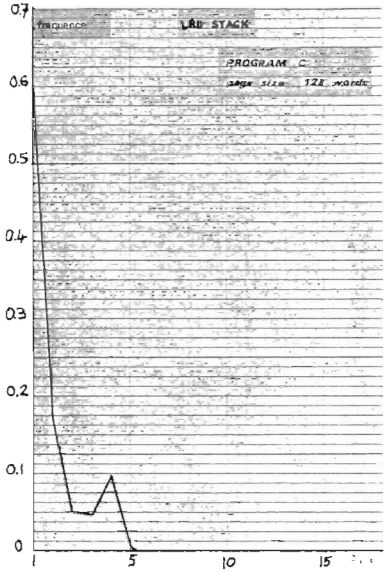
- in the first one, the information to be addressed consists of 128 words blocks (pages of 128 words). The block size has been chosen because of the MITRA 125 addressing mechanism with which only

128 words can be directly addressed (otherwise a base register has to be modified).

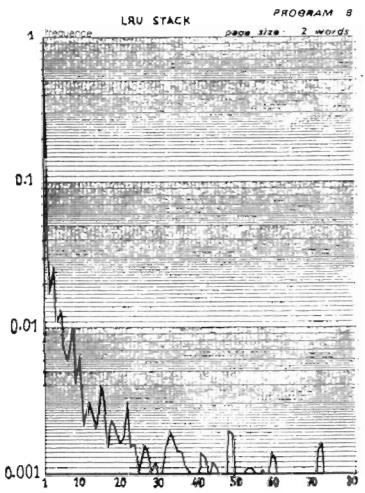
- in the second one, the information to be addressed consists of double word blocks (pages of 2 words). In this case, our purpose is to find out some "intimate" locality.



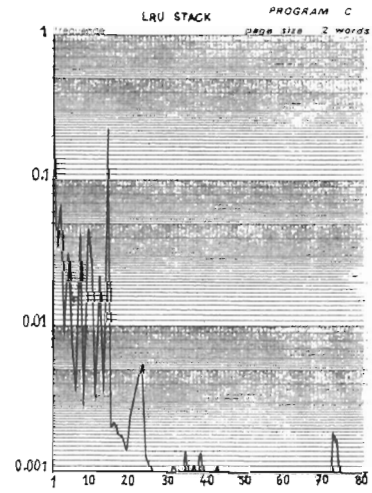
(3.1)



(3.2)



(3.3)



(3.4)

Figures 3 : LRU Stacks

We give in figures 3 some typical results we obtained. We have plotted the probability for a rank of the LRU stack to be referenced as a function of the rank.

As far as the 128 words stacks are concerned, we observe a global decrease (see figure 3.1) which can be locally contradicted on a few ranks (see figure 3.2). This fact comes from sudden cyclic locality changes. Despite that point, we may conclude there exists some locality phenomenon.

We may notice that such a behaviour is very close to the ones met on large computers which use other addressing mechanisms and more sophisticated instructions.

As for the double word stacks, the decrease is not so obvious because of the influence of program loops (see figures 3.3 and 3.4). This influence is particularly noticeable for programs A and C (we give in figure 3.4 the results for program C) where some ranks the order of which is greater than 10 or 40 have almost the same probability as the first rank to be referenced. It is noteworthy that some of these peaks can be interpreted to give an order of magnitude of program loop sizes. Note also that the high probability for the first rank to be referenced indicates that the prefecting mechanism described in section 2.1 will increase the hit ratio of the cache, since after having referenced a double word, there is a high probability for referencing it again.

3.2.2 - Working Sets [8]

The results we have just presented do not allow to size up the cache and do not give a detailed behaviour of programs with respect of memory. Therefore we have studied the working-sets [8] of the different programs. In fact, instead of the working-set itself, we have studied its size. Because of the MITRA 125 addressing mechanism (see § 3.2.1) we considered again 128 words pages.

We give in figures 4 two kinds of results. Both are related to program locality, but the second one can be used to roughly estimate the global size of the cache.

a - program locality

We have plotted the mean size of the working-sets as a function of their window $T(\bar{W}(T))$. The slower the increases of these curves, the more "local" the programs. Let us notice that these curves are unit slope straight lines for a program not "local" at all which accesses a new page at each reference. From figure 4.1, we can conclude that the notion of locality exists (on the figure, since the scales are quite different, a unit slope straight line is very "close" to vertical).

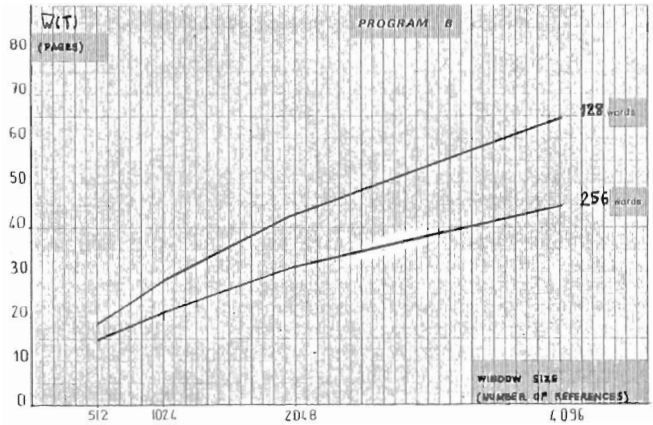
b - global size of the cache

The cache will be all the more efficient as its size will be large enough to load working-sets of programs. We give in figure 4.2 the working-set according to the time for a relatively "excited" program (program B). The working-sets under study have a window of 2048 references. Their sizes are counted in number of 128 words pages and the time is counted in steps of 512 references.

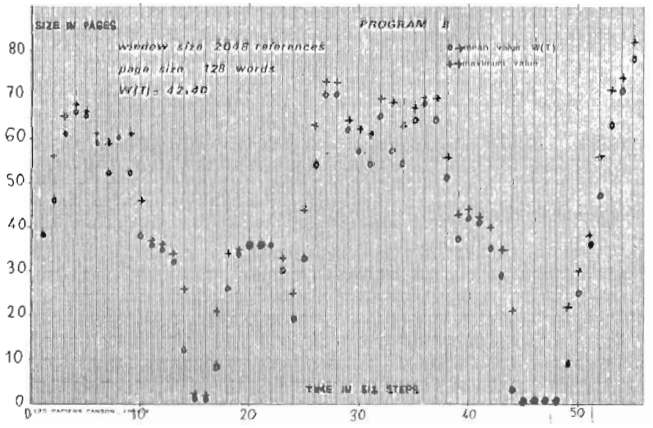
From this kind of figures of merit, we may deduce that

- a waiting loop has a working-set of roughly 5 pages,
- the "quiet" programs have working-sets of 20 to 30 pages and "excited" ones of 40 to 50 pages,
- no program has a working-set which contains more than 80 pages.

We can conclude that the order of magnitude of the cache size is possibly about 10K words. This will be corroborated by the results presented in section 4.



(4.1)

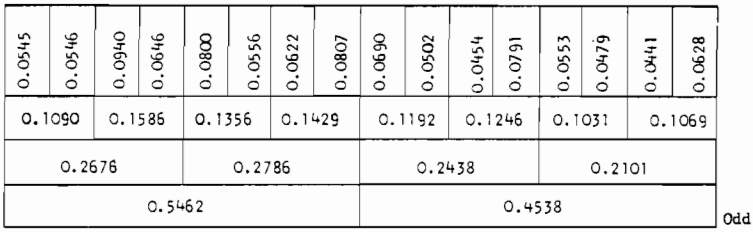


(4.2)

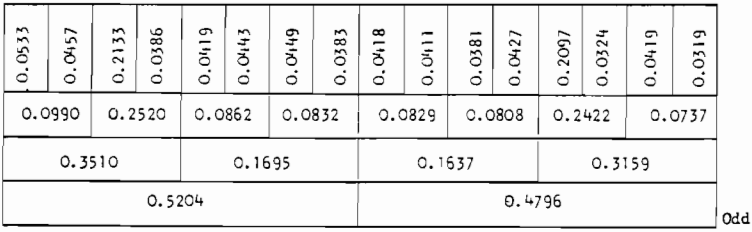
Figures 4 : Working Sets

3.3 - Program behaviour with respect of the low order bits of the referenced addresses

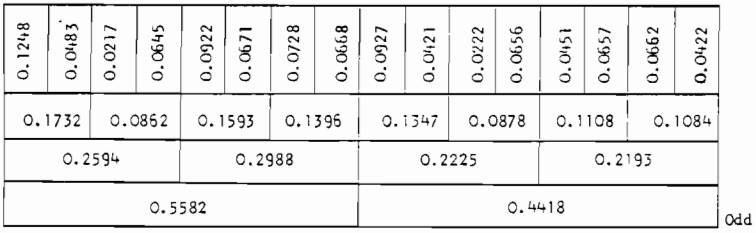
Because of the cache structure (see § 2.1, figure 1), the low order bits of a referenced address take a prominent part in the cache performance, from the standpoints of both hit ratio and segment space utilization. In particular, a good segment space utilization needs a uniform probability distribution of the low order part of the referenced addresses, since, in this case, all the segment entries have the same probability to be accessed. (Notice the conflict with the "locality" assumption).



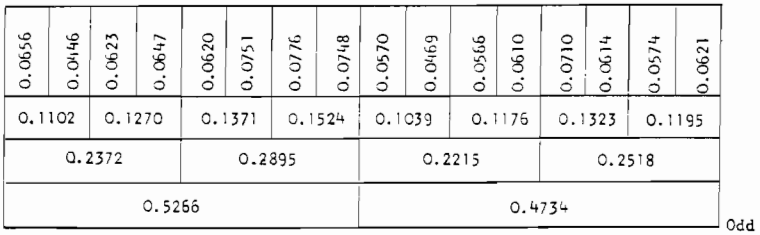
Program A



Program B



Program C



Program D

Figure 5

In this context, we have estimated histograms of the low order part addresses generated by programs. We have considered low order parts coded on 1, 2,...,10 bits. That is we have estimated the entry access probabilities of segments of $2, 2^2,...,2^{10}$ words.

The results we obtained (see figure 5) shows that these probabilities are not equal (non uniform distribution) for large segments (many entries). If the segments are small, the accesses are more balanced, but they can fluctuate according to the programs (see programs B and C). Therefore if the segments are too large, bad segment space utilizations have to be expected. This is corroborated by the results presented in section 4.

We give in figure 5 some results we obtained for segments of 2, 4, 8 and 16 entries. Notice that the probabilities to access odd or even addresses are relatively equal but the ones for the even addresses are always slightly greater.

3.4 - Conclusion

Up to now the results we have presented deal with the only program behaviour, but they permit to have some strong conclusions on the cache architecture :

- the idea of a cache memory is quite justified since the concept of "locality" exists both at a fine level (the 16 bits words) and at a more global one (the 128 words page).
- the cache will be large enough, about 10K words (the maximum working-set size).
- the segments will not be too large so that they are filled enough, but not too small in order to allow the loading of a "locality".
- a prefetching mechanism will increase the hit ratio of the cache (see § 3.2.1).

4 - THE PERFORMANCE EVALUATION

4.1 - The simulator

Precise cache performance has been evaluated by simulation. A simulator of the cache structure as described in section 2.1 was built in such a way that it permits to study the impact of :

- the cache size,
- the number of segments,
- the replacement algorithm (three algorithms were considered : FIFO, LRU and RANDOM),
- the policies concerning write accesses originated either from CPU or from i/o channels.

The inputs of the simulator consists of addresses'traces and the outputs provide information on hit ratios, space utilizations (global and for each segment), stationary or non-stationary behaviour of the cache (in case of stationarity, condidence intervals are built on the different figures of merit).

The results we seek for are of three kinds :

- to evaluate the minimum performance to be expected for the cache (which can be reached in the worst case).

- to study transient behaviour of the cache in order to find out reasonable performance (which can be expected, in average, on a long run).

- to evaluate the influence of the write accesses (generated from either CPU or i/o channels) on performance.

4.2.- The minimum performance

We have considered the worst operation conditions, that is :

- the cache is started empty. Our purpose is to take into account a sudden context change large enough so that information contained in the cache is no longer relevant.

- the length of the addresses' trace feeding a simulation run is limited to one block of 32K consecutive referenced addresses (see § 3.1).

- the blocks selected for the simulation runs are particularly "excited" in order to get pessimistic performance.

The approach we present aims at determining performance reachable at any time. In other respects, building confidence intervals in such conditions has the following meaning : all the conditions during the experiments are stationary. That is, intuitively speaking, the references are always rather not "local" and "complete" cache "renewals" regular and relatively frequent (in the average, every 32K references).

We give in figure 6 the characteristics of the "excited" blocks selected according to the program behaviour studies we have presented in section 3.

Programs	Selected block numbers (see Annexe I)	Numbers of references different within the block	Min. Numbers of references different for the whole program	Max. Numbers of references different for the whole program
A	16	3335	495	3335
B	21	5687	1848	7762
C	3	2544	74	4718
D	18	3550	1668	3639
E	17	4632	3176	8141
F	19	5874	3012	8134
G	17	9278	57	11019
H	8	3077	1350	8125

Figure 6

We present some results we obtained in figures 7. The figure of merit we consider is the hit ratio.

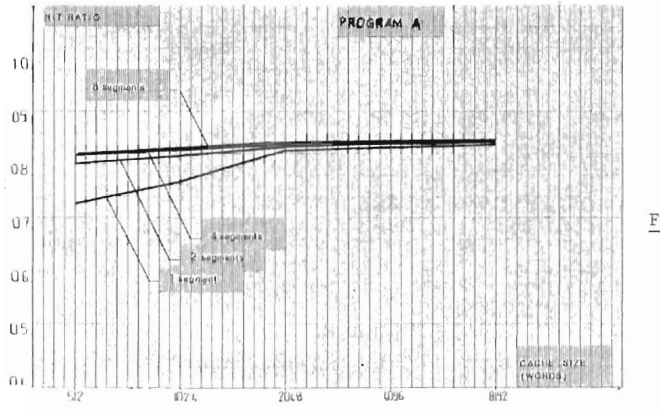


Figure 7.1

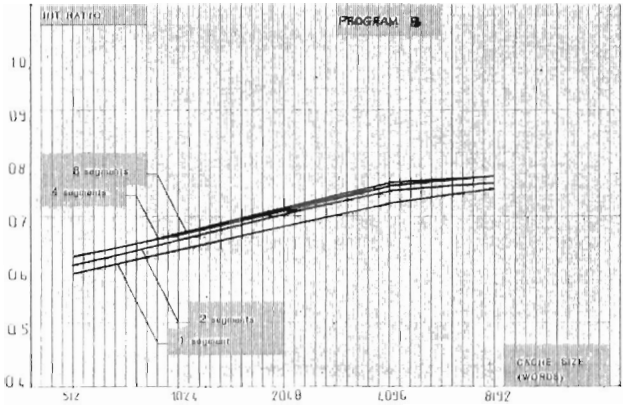


Figure 7.2

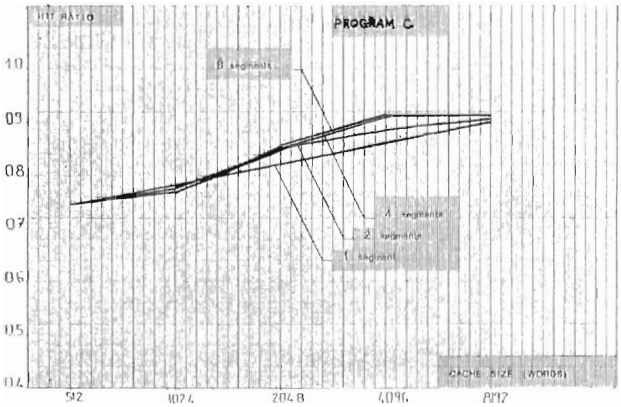


Figure 7.3

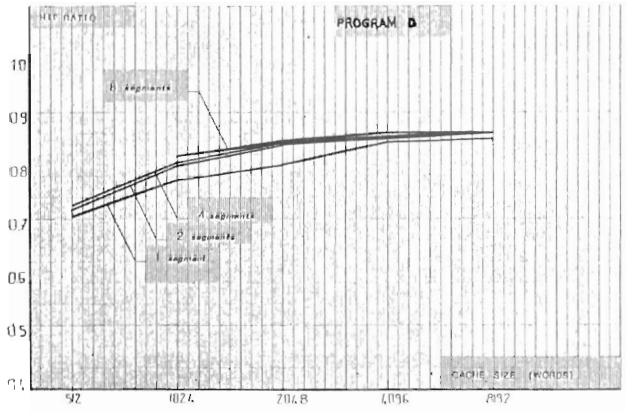


Figure 7.4

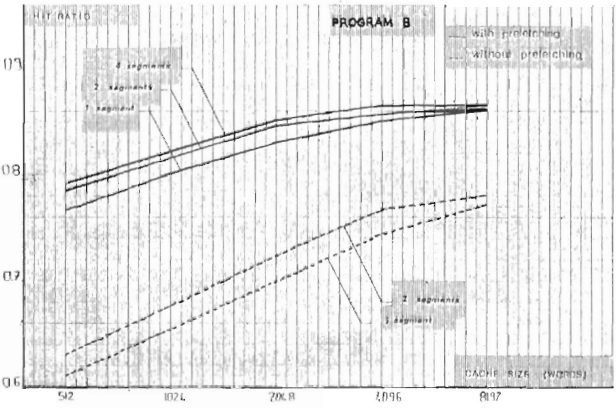


Figure 7.5

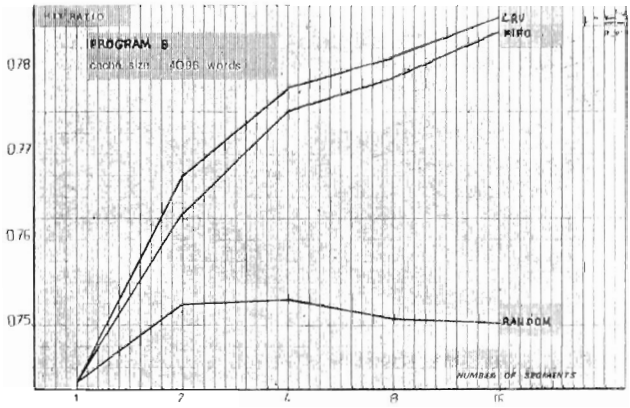


Figure 7.6

We can summarize these results as follows :

1 - Replacement algorithm

. the replacement algorithm has a slight influence on hit ratios. In general, the confidence intervals obtained are overlapped.

. the LRU algorithm is often the best and FIFO is very close to LRU. RANDOM is the worst.

. the influence of the replacement algorithm becomes noticeable when the cache is small and has many segments. That is normal.

2 - Number of segments

. the hit ratios increase with the number of segments.

. this parameter has a strong influence, if the cache is small.

3 - Global size of the cache

. the hit ratios increase with the cache size.

. this increase depends on the programs.

. beyond 8K or 10K, this parameter has a small influence on performance.

4 - Prefetching

. this parameter has a strong influence on performance.

. prefetching seems to be essential in order to get satisfactory performance.

All the results we present have confidence intervals the widths of which vary from $\mp$ 0.05 for the small caches (512 words) to $\mp$ 0.03 for the large ones (8K words).

4.3 - Transient behaviour of the cache

4.3.1 - Stationarity of time intervals between cache faults

The hit ratio we are interested in derives from time intervals between cache faults counted in numbers of references. Therefore we will consider these time intervals in order to get some results on the cache stationarity. Let T be the period during which we observed the cache and $t_1, t_2, \ldots, t_n \leq T \leq t_{n+1}$ the n instants at which n cache faults occurred. If we assume stationarity, we get (see [9]) :

$$\lim_{n\to\infty} U_n = \lim_{n\to\infty} \frac{\frac{\sum_{i=1}^{n} t_i}{n} - \frac{1}{2}T}{T\sqrt{\frac{1}{12n}}} = N(0,1) \qquad (5)$$

That is U_n converges towards a Gaussian distribution. Notice that the test we consider is optimal [9], if the stochastic process is Poisson with an arrival rate given by :

(6) $\lambda(t) = e^{\alpha+\beta t}$

In that case, we get :

$\beta = 0$ <=> stationarity

$\beta \neq 0$ <=> non-stationarity

We computed the U_n's for the programs A, B, and C for large n's by considering the whole traces without taking into-account the "holes" (see section 3.1). This approach seems reasonable if we consider the figures given in Annexe I where the "holes" do not disturb the sampling. The results we got shows that the U_n's absolute values are very much greater than 1.96, that is (see a table of Gaussian distribution) the stationarity is rejected with a 95 % confidence. Therefore stationarity does not exist for programs A, B and C as far as we consider them globally.

We repeated computations of U_n's for different n's (hence for different periods T) in order to find out some piecewise stationarity. This assumption has been rejected for the programs B and C where the stationary regions were very "small". On the contrary, for program A we found out "large" stationary regions (from 10,000 to 100,000 references) interrupted by "small" non stationary ones (from 1,000 to 5,000 references). Thus, strictly speaking, there does not exist a stationary state performance. Therefore we undertook a more precise study in order to formalize this notion.

For that purpose, we defined a new measure intermediate between traces given in Annexe I and working-sets, more appropriate to take into account working-set variations. The main memory is divided into 128 words pages and its "occupancy" is represented by a boolean vector where 0's or 1's indicate whether the corresponding pages were referenced or not during the considered time window. The measure we consider is the distance between two such consecutive memory "occupancies". That is :

(7) $d(\bar{x}, \bar{y}) = \sum_{i=1}^{n} x_i \oplus y_i$ where $\bar{x}$ and $\bar{y}$ are two memory "occupancies" and x_i, y_i, their respective components.

It is obvious that the distance we just defined is the number of the components of vectors $\bar{x}$ and $\bar{y}$ which differ. So the larger the distances, the more "excited" the programs.

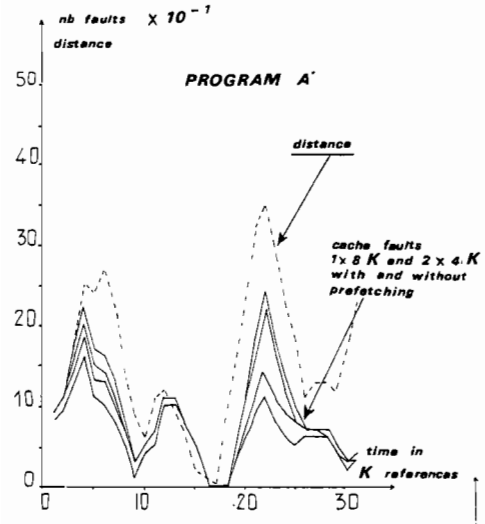
We give in figures 8 the results we obtained for programs A', B' and C' which are very similar to programsA, B and C reported through this paper. The time windows we considered are of 1024 references. The Y-axes represent both the distances (between two consecutive memory "occupancies") and the numbers of cache faults (on a time window), whereas the X-axes represent the time counted in windows of 1024 references. The dotted lines are distances and the continuous lines are numbers of cache faults for caches of :

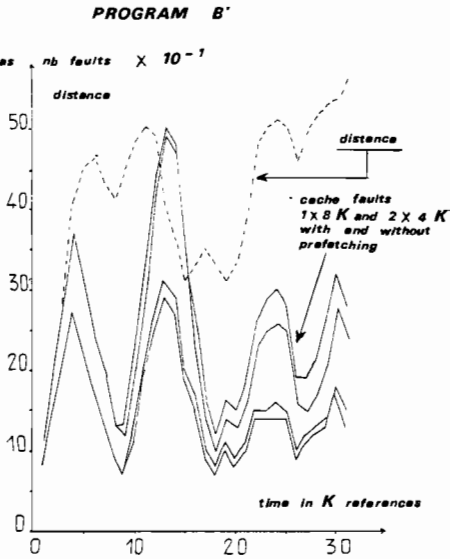
1 - 2 segments of 4K with prefetching

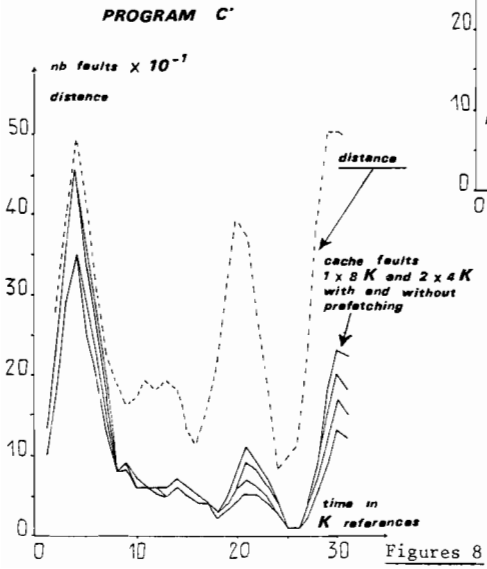
2 - 1 segment of 8K without prefetching

3 - 2 segments of 4K with prefetching

4 - 1 segment of 8K without prefetching.


nb faults x 10⁻¹
distance
PROGRAM A'
distance
cache faults
1 x 8 K and 2 x 4 K
with and without
prefetching
time in
K references
0
10
20
30
40
50


PROGRAM B'
nb faults x 10⁻¹
distance
distance
cache faults
1 x 8 K and 2 x 4 K
with and without
prefetching
time in K references
0
10
20
30
40
50


PROGRAM C'
nb faults x 10⁻¹
distance
distance
cache faults
1 x 8 K and 2 x 4 K
with and without
prefetching
time in
K references
0
10
20
30
40
50

Figures 8

When the curves are separated from each other, they always are in the order indicated above (1, 2, 3, 4). From figures 8, we can notice that the cache (in fact, its performance) immediately reacts to program behaviour : a mediocre performance corresponds to an "excited" behaviour (concordance of peaks) and a good one to a "quiet" behaviour (concordance of troughs). From this strong evidence, we tried to model the cache by a simple transfer function between distances representing program behaviour and cache fault rates representing the figures of merit under consideration. Unfortunately we were unable to find out such a function.

4.3.2 - The hit ratio as a function of time

From the previous results we have presented, we cannot consider a limiting quantity such as :

(8) $\hat{p} = \lim_{t \to \infty} \hat{p}(t)$ where $\hat{p}(t)$ is an estimation of the hit ratio computed between the instants 0 and t.

since there does not exist a steady state. Therefore we studied hit ratios as functions of time. We give in figure 9 some results we obtained. In order to get reasonable sizes for the different figures we used a logarithmic scale for the X-axes where the time is counted in references. For all the experiments, all the 32K references blocks of a program trace (see section 3.1) were considered as consecutive and devoid of "holes" (see also 4.3.1).

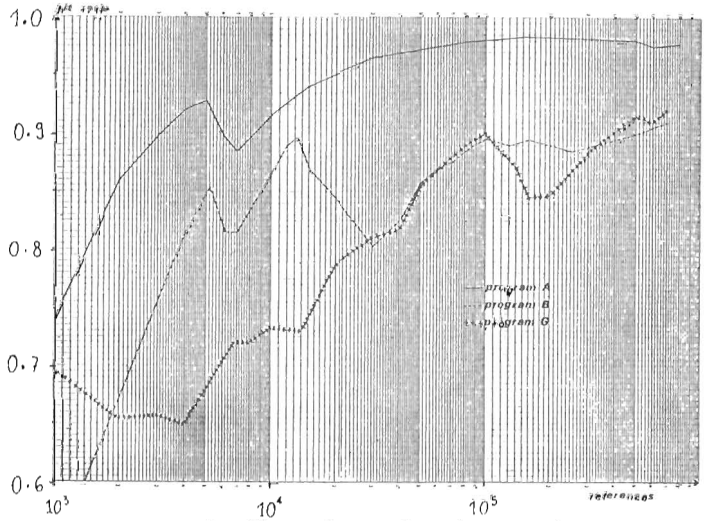


Figure 9 : Hit ratios as functions of time

From all the results we got, it seems there exists, except perhaps for programs B and G, an asymptote (a limiting hit ratio) located a bit beyond 0.9. This would lead to conclude that the limiting quantity in (8) exists. In fact we have to be careful because $\hat{p}(t)$ is an cumulative quantity and therefore not much sensitive to fluctuations. However that may be we give these limiting quantities for some caches in figures 10. To understand the behaviour of the cache as the time is running we give in figures 11 the space utilization of the cache as a function of time. Except for program B, which is very scattered in memory (see Annexe I), we notice that the space utilizations are very low. We must not deduce from this that the cache is badly used : it is the price we have to pay in order to get satisfactory performance.

Programs	LRU without prefetching	LRU with prefetching
A	0.977	0.979
B	0.910	0.954
C	0.972	0.984
D	0.984	0.991
E	0.900	0.949
F	0.903	0.946
G	0.877	0.922
H	0.936	0.960

Figure 10.1

Hit ratios of caches of two segments of 4K (confidence intervals less than 0.01)

Cache	LRU without prefetching	LRU with prefetching
1 × 8K	0.889	0.942
2 × 4K	0.910	0.954
4 × 2K	0.926	0.959
8 × 1K	0.934	0.961
1 × 4K	0.835	0.920
2 × 2K	0.858	0.934
2 × 1K	0.797	0.900
2 × 512	0.738	0.862

Figure 10.2

Hit ratios of different caches for program B (confidence intervals less than 0.01)

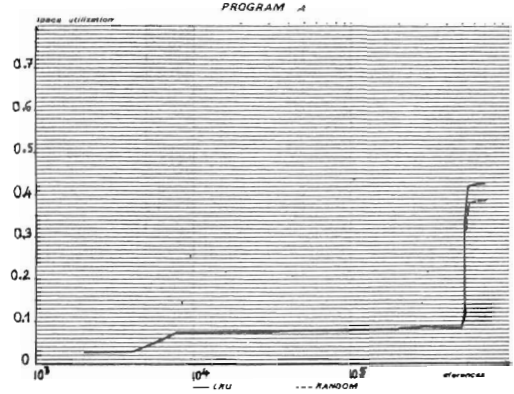


Figure 11.1

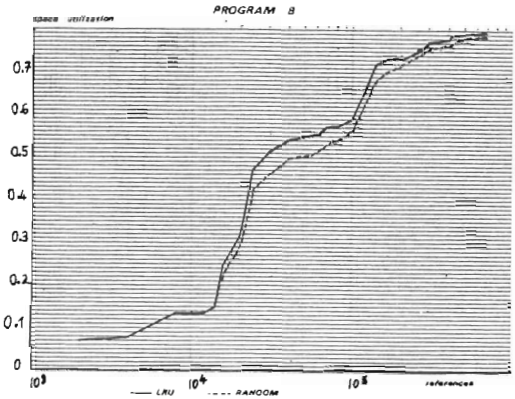


Figure 11.2

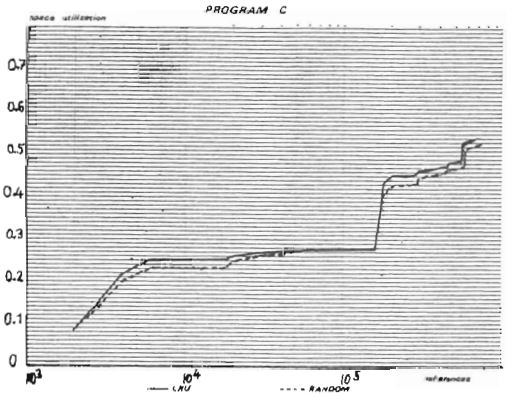


Figure 11.3

4.3.3 - The influence of the initial state

The results given in section 4.2 are surely pessimistic but the ones given just above are perhaps optimistic. Therefore we studied the influence of the initial state of the cache on the performance. In this view we give in figures 12 some results we obtained. We have drawn hit ratios as functions of time, on the one hand the simulation experiments were started with the cache empty of information and on the other hand the cache was initialized with the 32K references blocks preceding the selected one (see section 4.2).

From these experiments we may conclude that the average cache performance on a long run will be probably a lot better than the results reported in figures 7 and perhaps not so far from the ones given in figures 10.

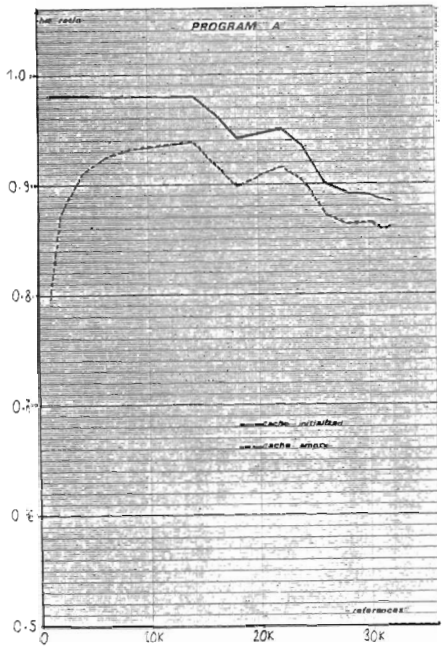


Figure 12.1

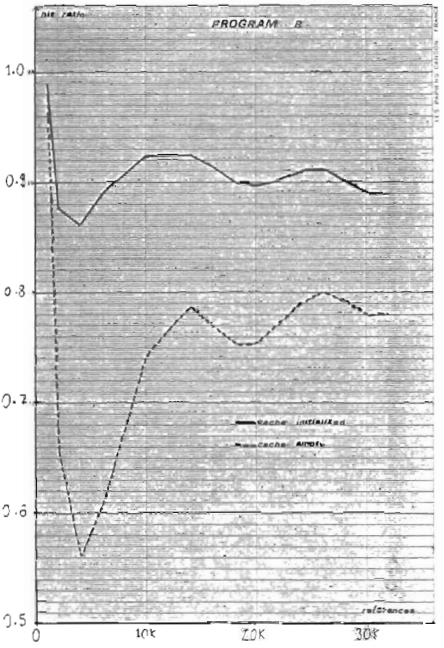


Figure 12.2

4.4 - The influence of the write accesses

A word in the cache can be read or written either by the CPU or by the i/o channels. Several policies can be implemented as far as the write accesses (modification of information) are concerned. Until now, the policy we have followed in all the experiments consists in :

1 - invalidating the data within the cache which are modified by i/o channels,

2 - not loading the data outside the cache which are modified either by the CPU or by the i/o channels (write accesses).

Programs	Cache	LRU without prefetching	LRU with prefetching	Total Number of analyzed references	Total Number of i/o accesses	Number of write i/o accesses
A	1 × 4K	0.978	0.991	1,245,184		
	2 × 4K	0.996	0.998			
	1 × 8K	0.978	0.995			
B	1 × 4K	0.868	0.953	1,295,419		
	2 × 4K	0.947	0.988			
	1 × 8K	0.925	0.977			
C	1 × 4K	0.944	0.984	1,306,624		
	2 × 4K	0.982	0.993			
	1 × 8K	0.977	0.988			
D	1 × 4K	0.967	0.988	980,507		
	2 × 4K	0.993	0.998			
	1 × 8K	0.984	0.995			
E	1 × 4K	0.817	0.913	1,132,956	25,033	13,924
	2 × 4K	0.944	0.978			
	1 × 8K	0.865	0.949			
F	1 × 4K	0.852	0.941	682,308	7,637	5,820
	2 × 4K	0.937	0.977			
	1 × 8K	0.904	0.966			
G	1 × 4K	0.815	0.935	672,005	19,780	16,123
	2 × 4K	0.925	0.976			
	1 × 8K	0.892	0.958			
H	1 × 4K	0.899	0.954	325,445	2,720	2,235
	2 × 4K	0.953	0.976			
	1 × 8K	0.923	0.966			

Figure 13

(Confidence intervals less than 0.01)

We will compare this policy with the new one that we consider now :

1 - the data within the cache which are modified by i/o channels are updated,

2 - the data outside the cache which are modified by the CPU (and not by the i/o channels (write accesses)) are loaded into the cache.

We give some interesting results we obtained in such conditions in figure 13. More details are presented concerning the i/o transfers of the programs E, F, G and H, since their percentages are higher than the others (see figures 2, section 3.1).

By comparing these results with the ones given in figures 10, we notice an increase of the hit ratios not negligible if we consider the speeding up of the target machine (see equation (4) and table in section 2.2).

5 - CONCLUSION

We have presented in this paper the performance improvement which can be expected by providing a mini-computer with a cache memory. The major criterion we have considered is the hit ratio from which we deduce a speeding up of the target machine. Our results show that in order to get satisfactory and stable performance a relatively large cache is needed, the size of which is of the order of 10K words. These results also show that when high performance is achieved, the improvements brought by implementing a prefetching mechanism or by increasing the number of segments are of the same order, but that in bad operating conditions a prefetching solution is always better. Therefore we may conclude that, in our case, we can easily triple the speed of the target machine with a reasonable trade-off.

In other respects, we carefully studied the behaviour of the program run on the machine and we showed the strong influence of the locality phenomenon on the performance.

ACKNOWLEDGEMENTS

We would like to thank Ph. LEBLANC and M. LEGOUX who participated in this work. In other respects we are indebted to SEMS for having entrusted us with this study and made it possible by supporting it and by providing us with basic data. In this regard, we would like to thank MM. MICHEL, BASTIE, DE LAMOTTE, GAULENE, GAUDEY, from SEMS.

REFERENCES

[1] LIPTAY J.S. - "Structural Aspects of the system/360 Model 85, II the Cache" - IBM Systems Journal, Vol. 7, n° 1, 1968.

[2] MEADE R.M. - "Design Approaches for Cache Memory Control" - Computer Design, Vol. 10, January 1971.

[3] KAPLAN K.R., WINDER R.O. - "Cache-based Computer Systems" - Computer, March 1973.

[4] SEMS - "MITRA 125 - Manuel de référence" - Brochures n° 4648 U1 FR and n° 4649 U1 FR edited by SEMS (Société Européenne de Mini-Informatique et de Systèmes), 1978.

[5] BELL C.G., CASASENT D. - "Implementation of a Buffer Memory in Mini-Computers"- Computer Design, Vol. 10, November 1971.

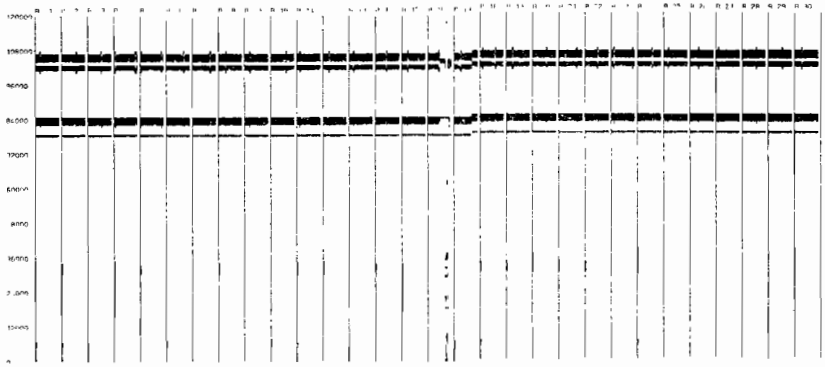
[6] BURGEVIN P., LEROUDIER J. - "Characteristics and Models of Program Behaviour"- National Conference ACM 76 - Houston (Texas) - October 1976.

[7] MATTSON R.L., GECSEI J., SLUTZ D.R., TRAIGER I.L. - "Evaluation techniques for storage hierarchies" - IBM System Journal n° 2, 1968.

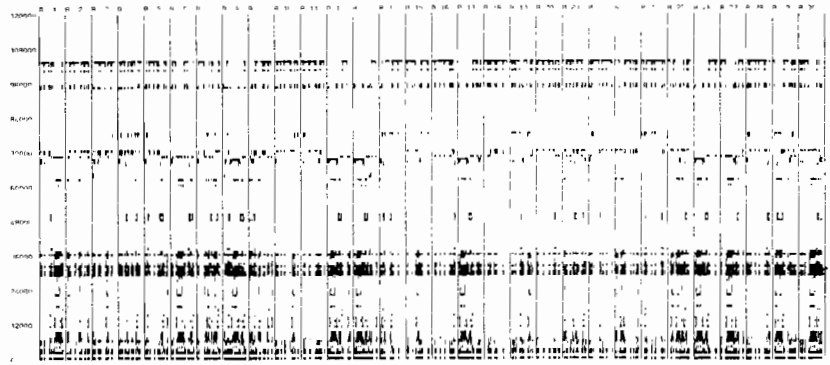
[8] DENNING P.J. - "The Working Set Model for Program Behavior" - Communication of the ACM, Vol. 11, n° 5, May 1968.

[9] COX D.R., LEWIS P.A.W. - "The Statistical Analysis of Series of Events" - Methuen - 1966.

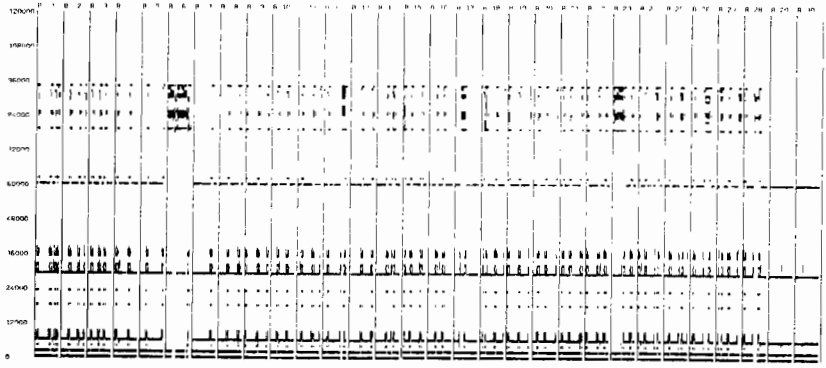
ANNEXE I



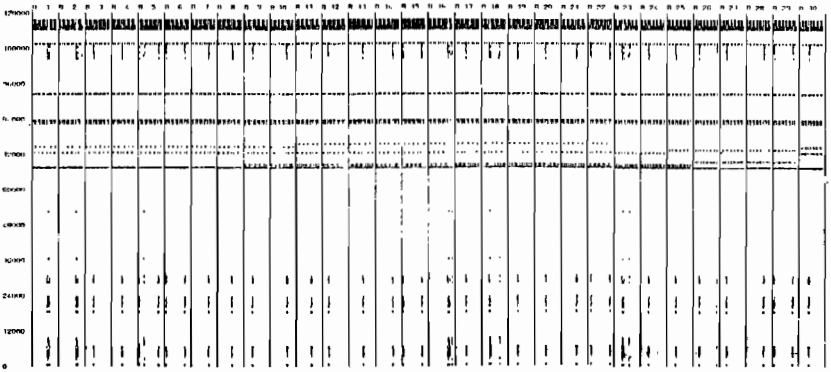
Program A



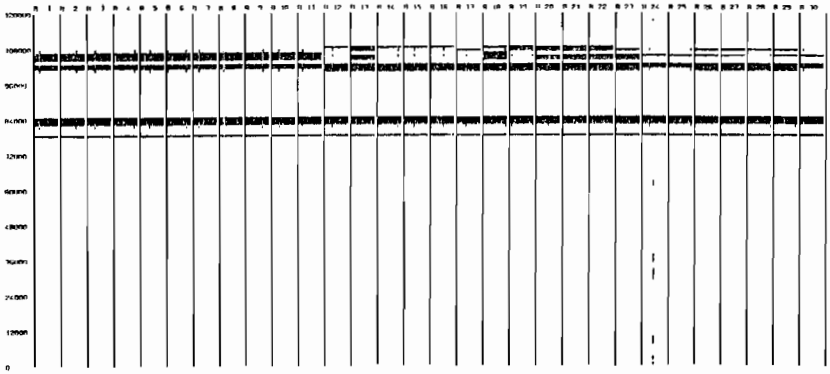
Program B



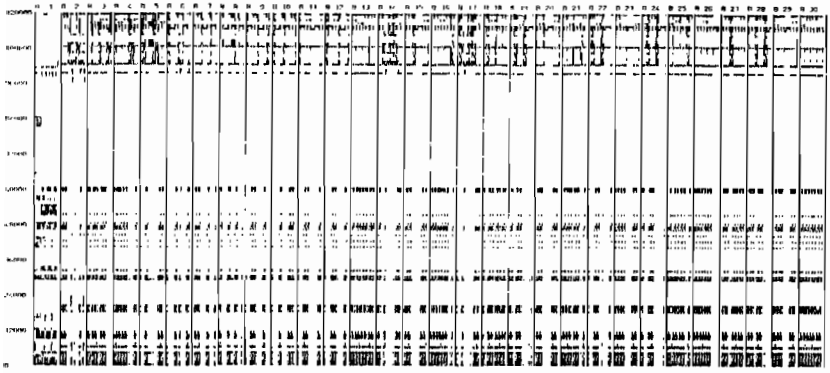
Program C



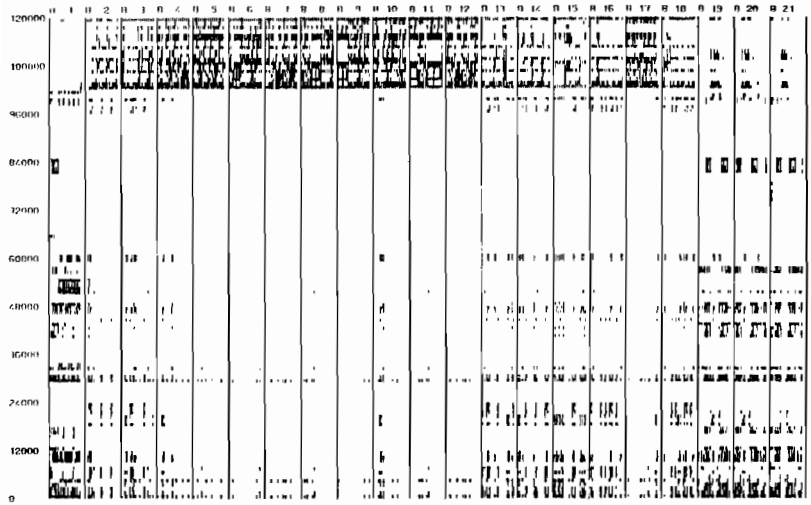
Program D



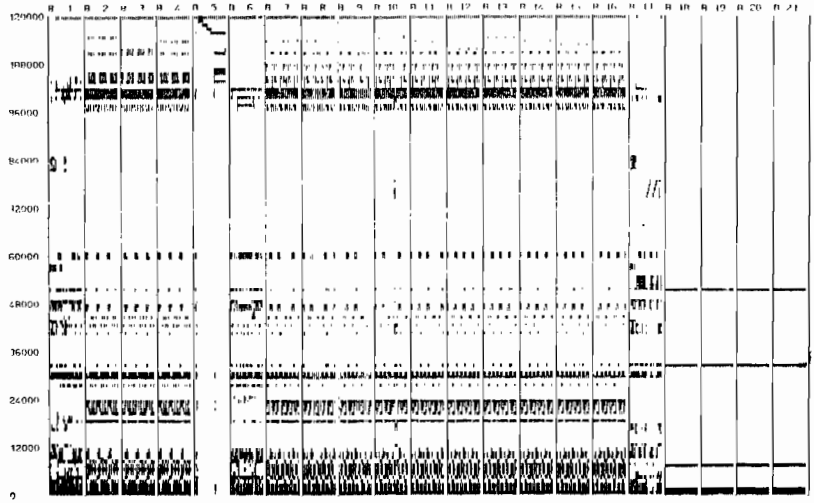
Program E



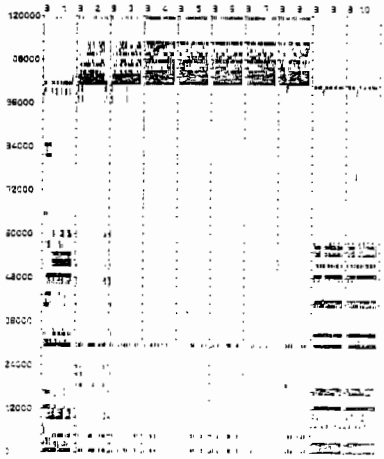
Program F



Program G



Program H



Program I

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

PERFORMANCE IMPROVEMENT BY FEEDBACK CONTROL OF THE OPERATING SYSTEM

A. Geck
Informatik Rechnerabteilung
Universität Karlsruhe
D-7500 Karlsruhe

Earlier tuning and optimization of the system (B6700) suggested that both the degree of multiprogramming and the intensity of memory usage should be dynamically adapted to configuration and load. A pragmatic approach was taken to do this by a controller integrated in the operating system and using feedback information. The controller was constructed using an intuitive model, implemented on the real system and tuned to maximize throughput. Synthetic mixes were used to measure performance and system behaviour under the original and the controlled system.

INTRODUCTION

The operating system of the B6700 offers the operator several system parameters to control the degree of multiprogramming and memory usage. In an earlier project /9,11/ the influence of these parameters on system behaviour were tested with various main memory sizes. It turned out that for a given load the set of optimal parameters depended on memory size. Tests also verified that for a given memory size the optimal values of parameters were depending on load characteristics as memory needs or CPU intensity. More precise analysis of the measurements of both experiments showed that the relationship between load demands for resources and resource capacities was most relevant.

At the B6700 installation in Karlsruhe, upgrading and temporary failure of single components had led to configuration changes. Measurements of the daily load had shown a great variance in CPU-, IO- and memory demands by users. To set the optimal values of parameters an operator many times a day might have adapted them to the actual load. The idea of this project was to enable the operating system to do this job.

THE REAL SYSTEM AT THE BEGINNING

The B6700 was the heart of a computer center managed by the University Karlsruhe Informatik Rechnerabteilung and therefore dedicated to teaching and research in Computer Science. There were up to sixty terminals connected to the system for interactive sessions. Batch jobs could be started by a terminal user or via a card reader.

The system had a dual processor,262 KW main memory, two disks and six packs. The operating system was a modified version of Burroughs Corporation Master Control Programm release II. 8.

Special characteristics of the operating system were:

- a virtual memory with segments of variable length,

- regulation of the degree of multiprogramming by a system parameter AVAILMIN representing a lower limit of available core which the system should not exceed,

- control of memory usage intensity by another system parameter OLAYGOAL defining the amount of overlayable memory to be overlayed to background per minute.

A SIMPLE MODEL FOR THE SYSTEM AND CONTROLLER

To understand system behaviour and to help construct the controller a system model was intuitively developed. This model made the following assumptions:

A1: The system has three main resources: CPU, IO and memory

A2: Each task demands service from each main resource. The amount of service asked for and the relationship between demands for different resources varies widely in time, and from task to task. The average arrival rate of tasks and the average distribution of demands among resources are not constant.

A3: For each resource there exists a lower and an upper bound to the load (user and system), which should not be exceeded to achieve optimal system performance.

A4: Load may be transferred from one resource to another (within certain limits) by changing system parameters and strategies. Load should be dynamically balanced in this way to achieve optimal system performance.

For the given real system, assumption one was apparently true. For other systems it may be necessary to include a special software resource, such as for example a data base system, or to divide IO into different resources for paging and user IO. It may also be useful not to include the CPU resource, if it is never a bottleneck.

Assumption two is surely valid for every environment with interactive users and therefore also for the modelled system. There are many reasons for load changes within more or less short periods of time in other environments.

In assumption three the existence of a lower bound is equivalent to the fact that each system resource should be kept busy to achieve good throughput. The existence of an upper bound for memory load is justified by the overhead required to manage a heavily loaded memory. An overloaded CPU with many waiting tasks causes no immediate overhead. But all these tasks need memory. Even if memory is not yet overloaded, memory management causes overhead, which normally increases with the amount of memory used. Therefore, if the CPU is heavily loaded, the probability of improving throughput by a larger load is low and limited, but savings in memory management overhead by limiting load are quite certain. That's why the model does not regard the highest possible load as the best load.

Load limiting is also necessary if optimum performance requires low or limited response times. The exact values of the bounds in assumption three may depend on system structure and the relative weighting factor for throughput and response times in the performance criterion. A high weight for throughput corresponds to higher values, a high weight for response times to lower values.

Assumption four establishes the possibility of partially transforming demands for service on one resource into demands for service on other resources. With a virtual memory system, it is easily seen how memory demands are transformed into CPU- and IO-demands, thus enlarging the virtual memory capacity. Load may be balanced

in such a system by controlling the degree to which such transformation takes place. For the B6700, this could be done by using the parameter OLAYGOAL; for other systems it may be done by controlling the frequency with which the usefulness of pages is tested, or by controlling the timelimits defining the working set of a task. Transformation of load is not only possible between memory and the resources CPU and IO. Many operating systems use complicated optimizations in IO-handling. This is a partial transformation of IO-demands into CPU-demands.

Assumption four requires load balancing by means of transferring load from the actual performance limiting bottleneck to other resources. Load balancing by selecting tasks during scheduling is not proposed, because task demands may be unbalanced over longer periods of time. In addition there may be not enough tasks waiting for selection or the need for short response times may prohibit task selection. Selecting tasks for load balancing, implies adapting the load to the system configuration, whereas changing parameters or strategies for load balancing, implies adapting system management to the load.

IMPLEMENTATION OF THE CONTROLLER IN THE REAL SYSTEM

The degree of multiprogramming was chosen to control overall load. Load may be increased by admitting new tasks to the system or by reactivation of suspended tasks. Load may be decreased by suspension of some task (the task loses all overlayable memory and must wait for reactivation).

Changing the OLAYGOAL parameter was used for load balancing. With this method load may only be transferred to or from memory. Memory management was that part of the system which had been well studied in earlier projects and apparently had a great influence on load distribution and performance. Other feasible methods of load transfer between resources were not included because of limitations in time and manpower.

The B6700 operating system realizes some sort of look ahead paging out: every three seconds a special task is activated that overlays segments chosen by a random strategy out of core. OLAYGOAL defines the percentage of overlayable core to be overlayed in advance within one minute. This strategy ensures a constant level of memory usage intensity: segments not referenced over a longer period of time are eventually overlayed.

A higher value for OLAYGOAL, shifts load from memory to CPU and IO (or more precisely, to CPU and paging devices and channels, which, under the B6700 operating system, are not physically separated from file devices and channels). A lower value shifts load in the opposite direction.

Figure 2 shows in principle, in which directions load is shifted by the different activities. By combining several activities, a wide range of possible load changes becomes available.

Based on these assumptions a model controller was developed. Is was assumed that a measure for the load of each main resource exists. A scale for this was divided into three regions by bounds corresponding to those defined in assumption three.

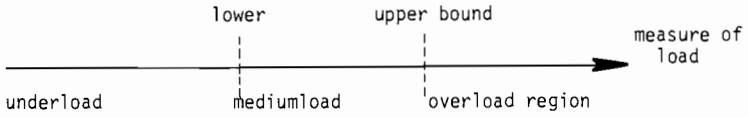


Figure 1

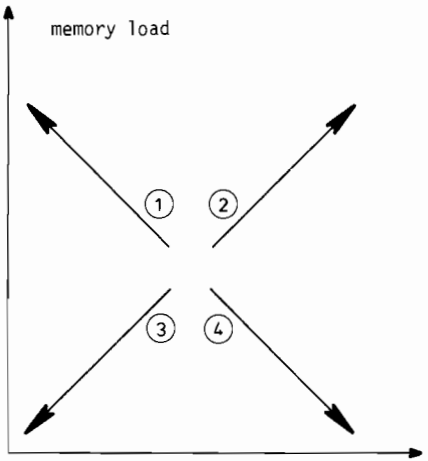


Figure 2:

(1) Decreasing OLAYGOAL

(2) Admitting new or suspended tasks

(3) Suspending tasks

(4) Increasing OLAYGOAL

Overall load was controlled by the following rules:

- Load is increased whenever all resources are underloaded,
- load is decreased whenever one resource is overloaded.

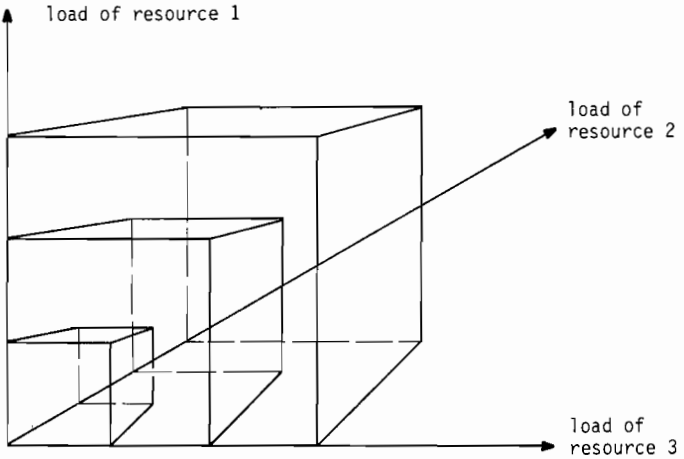


Figure 3:

In figure 3 the space of overall system load - like that of one resource - is divided into the three regions of underload, medium load and overload. The underload region equals the smallest cube, the medium load region equals the medium sized cube minus the smallest cube, and the overload region equals the largest cube minus the medium sized one.

The load balance was controlled by an additional rule:

- Load is transferred from the most heavily loaded resource to other resources.

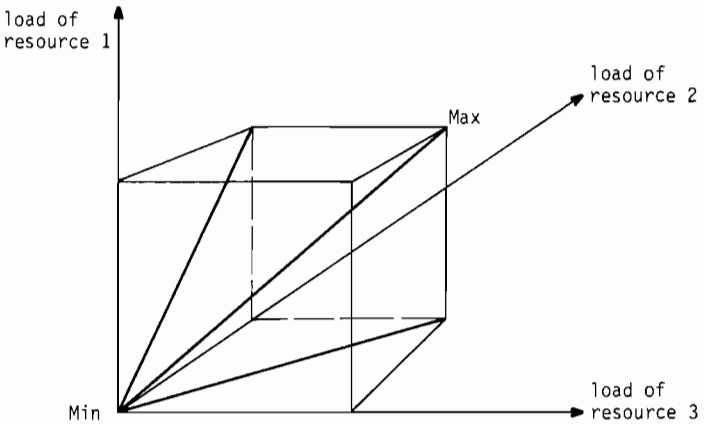


Figure 4:

In figure 4 the line crossing the cube from Min, the point at which all resources are unloaded, to Max, the point at which all resources are fully loaded, represents the region of balanced load. The three pyramids around it with their top in Min and their base equal to the upper, rear or right surface of the cube represent the regions within which load has to be transferred from resource one, two or three respectively to other resources.

To determine the memory, CPU and IO load eight terms measured by the operating system were used:

- AVAILABLE CORE, CC-OVERLAYS and PRESENCEBITS to define memory load,

- PROCESSTIME, ALIVE TASKS and ALIVE TASKS/MIXCOUNT to define CPU load,

- IOQUEUES of PACKS and IOQUEUES OF DISKS to define IO load.

AVAILABLE CORE is the amount of free memory, CC-OVERLAYS is the number of core to core overlays per second (performed if the operating system tries to overlay small segments in order to find space for large segments) and proved to be most sensible

to thrashing. PRESENCEBITS denotes the number of segments loaded after presence bit interrupts per second. PROCESSTIME is the percentage of time processors are busy, ALIVETASKS is the number of tasks waiting for a CPU or at a CPU, MIXCOUNT is the total number of tasks.

To compute a measure for memory, CPU and IO load respectively, the scale for all terms was divided into three regions. The scale for AVAILABLE CORE was further divided into six subregions. The resulting 64 different memory-, 27 CPU- and 9 IO-situations were mapped via tables into an underload, medium load and overload region for memory, CPU and IO respectively. Memory load regions were composed of ten subregions. A subregion reflected the relationship between the amount of memory used and the intensity of memory usage. Memory load definition was more complicated, because it had to be taken into account, whether intensity of memory usage could really be further increased or decreased by change of OLAYGOAL.

Using the rules established for the model controller, mapping of the actually measured memory, CPU and IO load into the possible activities was performed as follows:

M : If memory, CPU and IO were underloaded, new tasks were admitted or reactivated,

D : if either memory, CPU or IO was overloaded, one task was suspended,

Ø : if memory load was very low (most probably CPU and/or IO are then more heavily loaded), OLAYGOAL was set to zero,

- : if memory load was low or medium and memory usage intensity relatively high, OLAYGOAL was decreased,

+ : if memory load was medium or high, memory usage intensity relatively low, and CPU and IO were underloaded, OLAYGOAL was increased.

The exact values of the bounds of each measured term were controller parameters that could be dynamically changed. They were tuned to maximize throughput, as were parameters of the controller defining the range of possible values of OLAYGOAL and how fast OLAYGOAL was increased or decreased.

The controller was implemented as an operating system subroutine activated once a second. This was easily done, because the operating system of the B6700 was written in a high level language, ESPOL.

memory load

CC-Overlays			Low			Medium			High		
PRESENCEBITS			L	M	H	L	M	H	L	M	H
AVAILABLE CORE	L	1	H-10	H-10	H-10	H-10	H-10	H-10	H-10	H-10	H-10
		2	M-9	M-9	M-8	M-9	M-9	M-8	H-10	H-10	H-10
		3	M-9	M-9	M-6	M-9	M-8	M-6	M-8	M-8	M-6
		4	L-4	L-3	M-6	L-3	L-3	M-6	M-7	M-7	M-6
	M	5	L-2	L-2	M-6	L-2	L-2	M-6	M-6	M-6	M-6
	H	6	L-1	L-1	M-5	L-1	L-1	M-5	M-5	M-5	M-5

CPU-load

ALIVE TASKS		Low			Medium			High		
ALIVE TASKS / MIXCOUNT		L	M	H	L	M	H	L	M	H
PROCESSTIME	Low	L	L	L	L	L	L	L	L	L
	Medium	L	L	L	L	L	M	M	M	M
	High	M	M	M	M	M	H	H	H	H

IO load

IOQUEUES PACKS	Low			Medium			High		
IOQUEUES DISKS	L	M	H	L	M	H	L	M	H
	L	M	H	M	M	H	H	H	H

Activities

CPU load			Low			Medium			High		
IO load			L	M	H	L	M	H	L	M	H
memory load	L	1	M,Ø	Ø	D,Ø	Ø	Ø	D,Ø	D,Ø	D,Ø	D,Ø
		2	M,-	-	D,-	-	-	D,-	D,-	D,-	D,-
		3	M		D			D	D	D	D
		4	M,+		D			D	D	D	D
	M	5	Ø	Ø	D,Ø	Ø	Ø	D,Ø	D,Ø	D,Ø	D,Ø
		6	-	-	D,-	-	-	D,-	D,-	D,-	D,-
		7			D			D	D	D	D
		8			D			D	D	D	D
		9	+		D			D	D	D	D
	H	10	D	D	D	D	D	D	D	D	D

The interface to other operating system parts was established by variables and minor changes in other modules. To improve the influence of the OLAYGOAL parameter, it was necessary to change computation of the OS overlayable core to be overlayed in advance.

To avoid instability of the controlled system, activities were performed with caution. For example not more than one task was suspended every three activations of the controller. The controller was slowed down, thus enabling the operating system to react between two activations of the controller, and measured terms were smoothed.

As measurements showed, the demands of the controller itself were low. Within elapsedtime of 100 seconds the controller used less than 3 seconds IO time and 4 seconds CPU time, including times for the overlay of segments to background controlled by OLAYGOAL.

RESULTS

To measure performance changes caused by implementation of the controller, four synthetic mixes with different characteristics were run twice under the original and twice under the controlled system. The mixes consisted of batch tasks only and were based on the real load of some day. Performance was measured as throughput:

Performance = Number of tasks / (Finishtime of last - starttime of first)

Performance improvements by the controller varied between 8 % and 18 % for a single run. In the following table the average performance improvement for each mix is shown.

MIX	Finish of last-starttime of first (sec)		Performance (tasks/min)		Performance improvement
	original OS	controlled OS	original OS	controlled OS	
CPU-bound	1775	1588	6.73	7.52	12 %
IO-bound	1789	1613	6.73	7.40	11 %
slightly memory-bound	1985	1820	6.02	6.56	9 %
memory-bound	2268	1983	5.26	6.02	14 %

The original system with a constant value of 3 % for OLAYGOAL was best adapted to a slightly memorybound load, that is why performance improvement is smallest in that case. It was not well adapted either to CPU- or IO-bound loads, for which a lower value of OLAYGOAL is suitable, or to memory-bound loads, for which a higher value of OLAYGOAL is necessary. With all mixes the load limiting feature of the controlled system avoided unnecessary overhead.

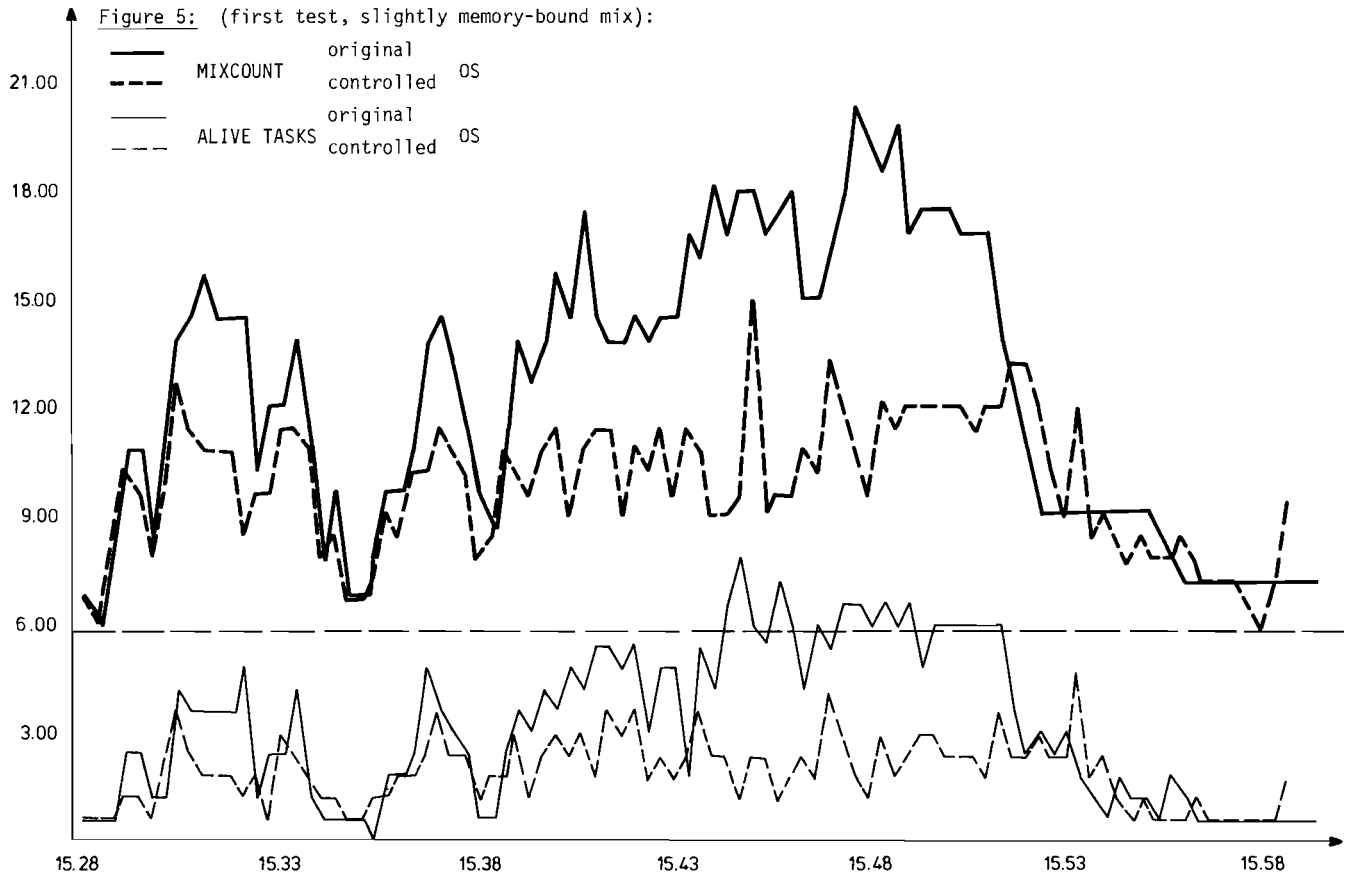


Figure 5: (first test, slightly memory-bound mix):
MIXCOUNT
original
controlled
OS
ALIVE TASKS
original
controlled
OS
21.00
18.00
15.00
12.00
9.00
6.00
3.00
15.28
15.33
15.38
15.43
15.48
15.53
15.58

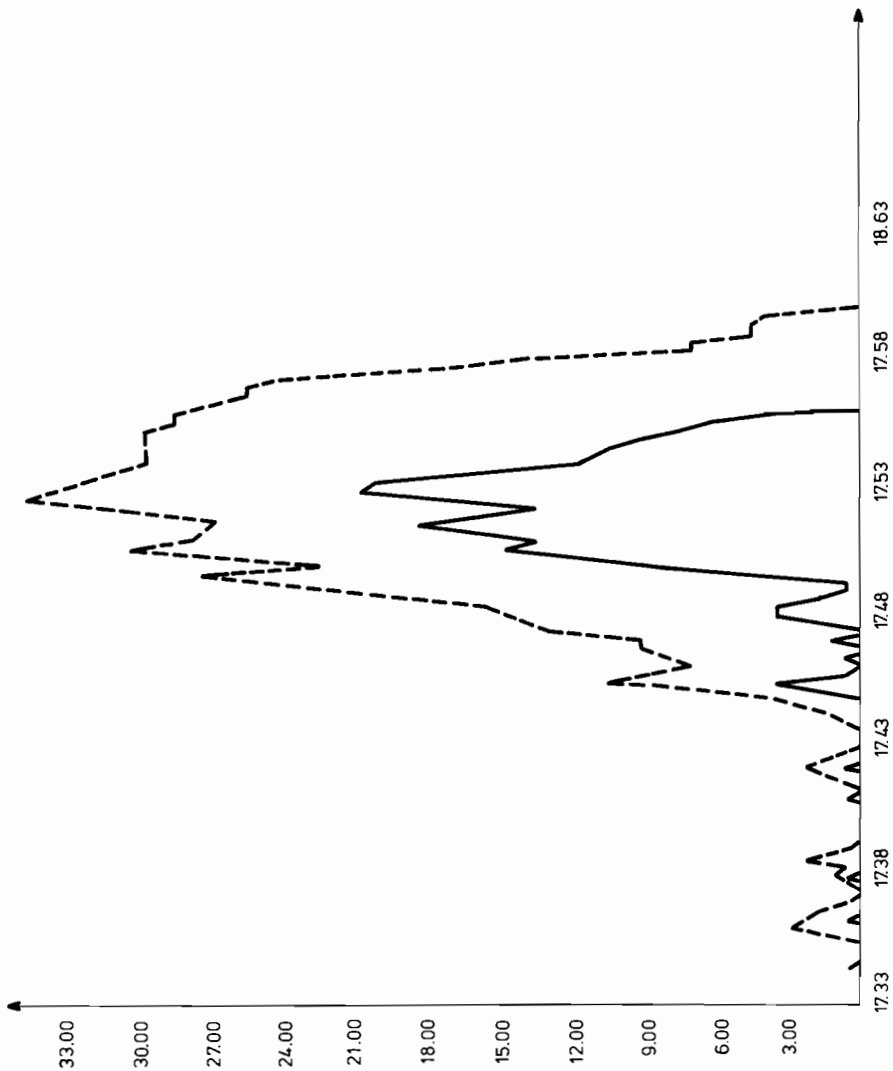


Figure 6 (first test, slightly memory-bound mix):
Time series for SCHEDULED TASKS
—— original OS, – – – controlled OS.

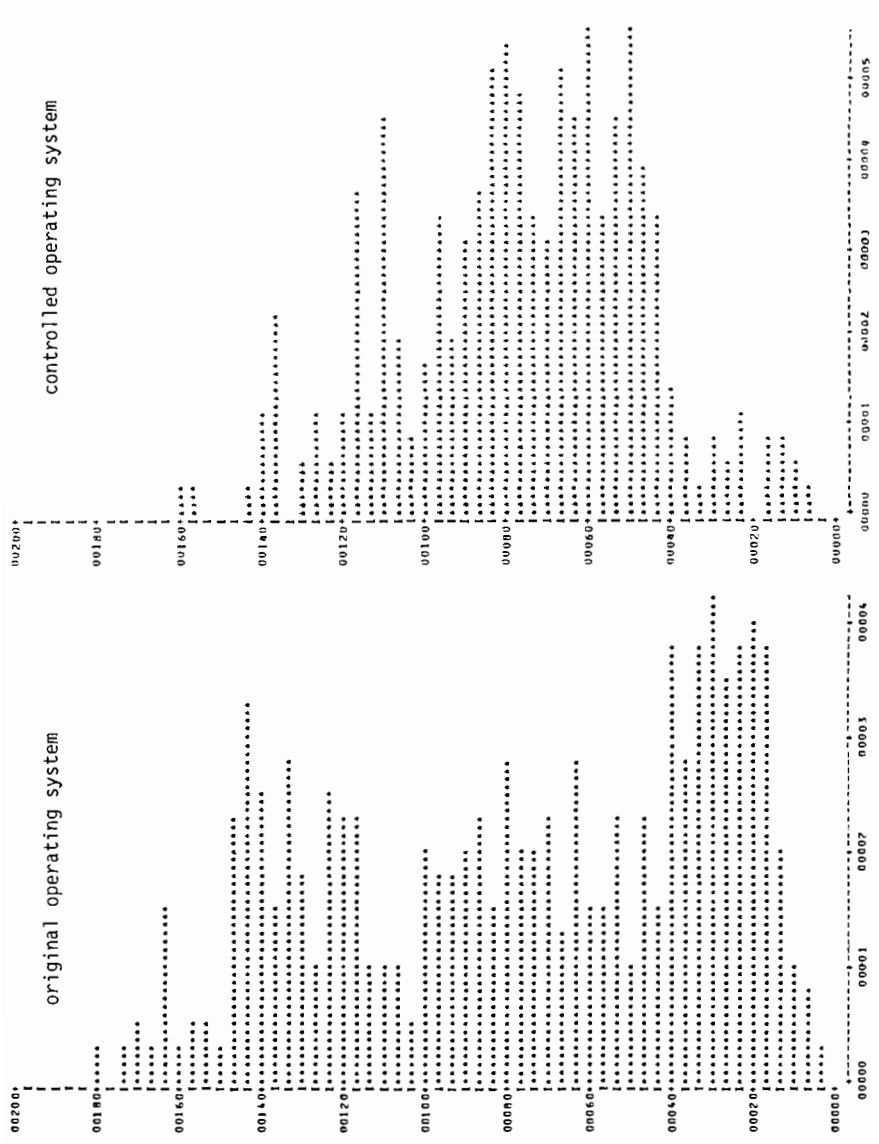


Figure 7 (first test, all mixes): AVAILABLE CORE distribution.

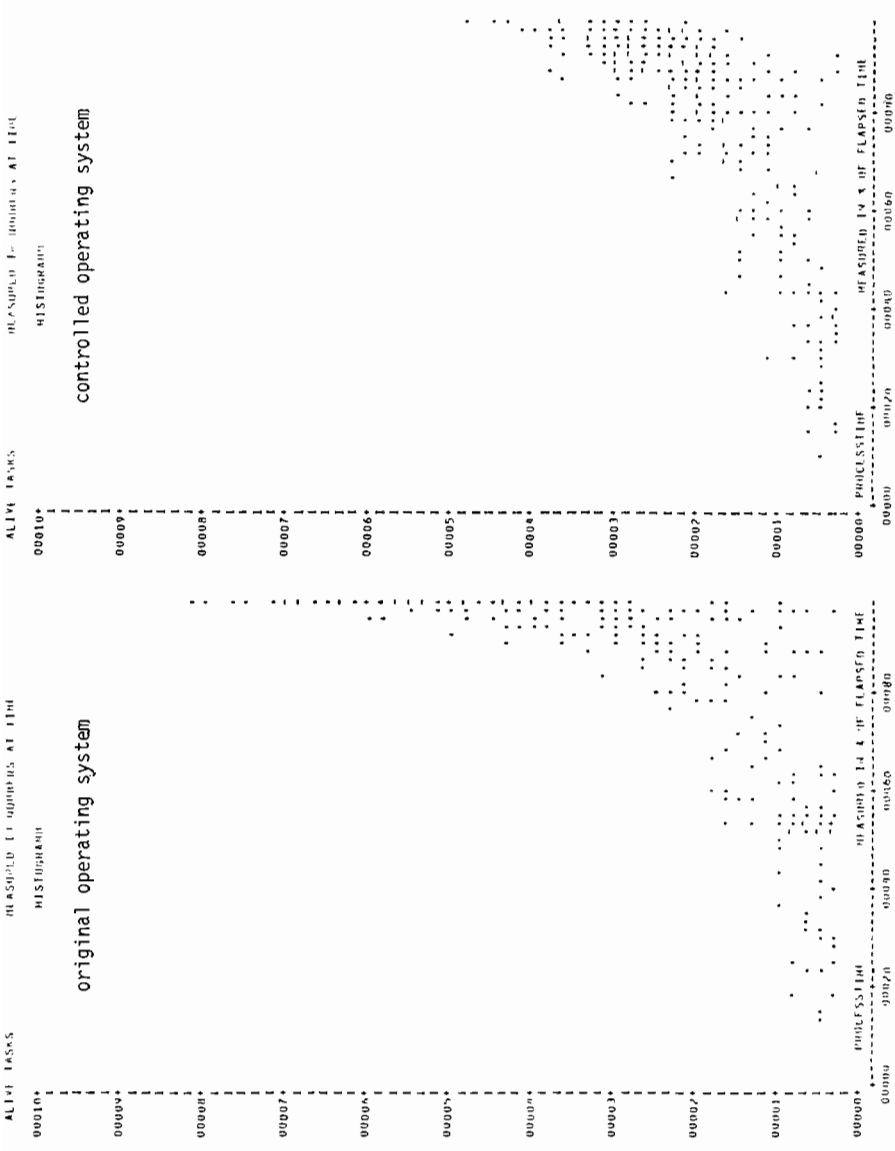


Figure 8 (first test, all mixes): PROCESSTIME/ALIVE TASKS Correlation.

The following diagrams show in more detail the differences in system behaviour under both operating systems. In figure 5 MIXCOUNT and ALIVE TASKS are outlined. The figure shows lower values for both MIXCOUNT and ALIVE TASKS under the controlled system thus demonstrating the load limiting feature of the controlled system. ALIVE TASKS values up to 8 under the uncontrolled system indicate CPU overloading (the system has 2 CPU s).

Figure 6 shows SCHEDULED TASKS, the number of tasks waiting to enter the Mix. Higher SCHEDULED TASKS values of the controlled system prove again the load limiting feature.

Figure 7 compares AVAILABLE CORE distributions. Under the original system the distribution has two peaks, one for a dangerously low value, caused by memory overloading, and one for a relatively high value, caused by unnecessary advance overlays by a constant OLAYGOAL value. Under the controlled system overloading is avoided and OLAYGOAL is adapted to memory situation thus producing a distribution with one peak for a low, but not dangerously low value.

In figure 8 finally the correlation of PROCESSTIME to ALIVE TASKS is plotted. Under the uncontrolled system quite a lot of measured points are found in the CPU overload region. Under the controlled system points have been shifted out of the overload region to a high load region.

REFERENCES

1. Alderson, A., Lynch, W.C., and Randell, B.(1972). Thrashing in a Multiprogrammed Paging System, Operating Systems Techniques, A.F.I.C. Studies in Data Processing No. 9. Academic Press, London · New York 152 - 167.

2. Brawn, B.S., and Gustavson, F.G. (1968). Program Behavior in a Paging Environment, AFIPS FJCC, vol. 33, part 2. 1019 - 1032.

3. Burnette, W.A., and Wright, L.S. (1976). An Approach to Evaluating Time Sharing Systems: MH-TSS A Case Study. ACM Sigm. Perfm. Eval. Rev., vol. 5, No. 1. 8-28.

4. Cantrell, H.N., and Ellison, A.C. (1968). Multiprogramming System Performance Measurement and Analysis. AFIPS SJCC, vol. 32. 213 - 221.

5. Denning, P.J. (1968). The Working Set Model for Program Behaviour. CACM 11. 323 -333.

6. Denning, P.J. (1968). Thrashing: it's Causes and Prevention. AFIPS FJCC, vol.33 part 1. 915 - 922.

7. Geck, A. (1977). Ansätze zur Regelung des Systemverhaltens der B6700. Thesis, University Karlsruhe, Germany.

8. Kleinrock, L. (1975).Queueing Systems. John Wiley & Sons, New York · London · Sidney · Toronto.

9. Mund, E. (1975). Messung und Optimierung des Systemverhaltens der B6700. Thesis, University Karlsruhe, Germany.

10. Spies, P.P. (1974). Struktur einer Rechenanlage mit der Fähigkeit zu adaptivem Verhalten, Rechnerstrukturen und Betriebsprogrammierung, Lecture Notes in Computer Science 13. Springer, Berlin · Heidelberg. 104 - 119.

11. Zorn, W. (1976). Systemoptimierung mit Hilfe von synthetischen Belastungsprofilen, Betrieb von Rechenzentren. Springer, Berlin · Heidelberg. 255 - 269.

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

A QUEUEING MODEL OF A TIMESLICED PRIORITY DRIVEN TASK DISPATCHING ALGORITHM

P.S. Kritzinger, A.E. Krzesinski and P. Teunissen
Department of Computer Science
University of Stellenbosch
Stellenbosch 7600
South Africa

ABSTRACT. The paper presents a queueing analysis of a pre-emptive, priority driven, timesliced dispatcher algorithm, typical of those found in many timesharing systems. A distinctive feature of the system being modelled is that a pre-empted task, when re-admitted to the dispatcher, is re-allocated a full timeslice rather than the residual. The combination of pre-emption, timeslice renewal and class dependent arrival and service rates, places the analysis beyond that of the well-known FB_N algorithms. The model is used to predict the behaviour of the dispatcher algorithm under various workloads.

1. INTRODUCTION

The Reference-Model Parameter Adaptive (R-MPA) control system [2] is a control system currently being developed to optimise the performance of a UNIVAC 1110 computer. The overall goal of the R-MPA system is to monitor workload processing and to adjust the operating system to respond optimally to any significant change in workload composition. The R-MPA system consists of three hierarchically inter-dependent models of the UNIVAC 1110 computer, namely a multiclass network model describing overall system performance and two detailed analytical models describing CPU dispatching and memory allocation respectively. This paper describes the dispatcher model, outlines its mathematical structure and presents an evaluation of dispatcher performance, namely average response times, queueing delays, queue lengths and CPU utilisation for each of the batch, timesharing and systems overhead fractions of the workload, as a function of workload composition, workload level and timeslice length.

2. THE DISPATCHER

The function of a dispatcher algorithm is to allocate CPU processing time so as to achieve maximum throughput for higher priority tasks while ensuring that lower priority tasks receive an adequate fraction of the CPU processing capacity.

The UNIVAC 1100 Executive (EXEC-8) classifies the workload into several distinct workload types, each type having a distinct priority. The dispatcher itself is based upon a pre-emptive, priority driven, timesliced multi-level feedback queueing system and is typical of multitask dispatchers in general. The feedback queueing system consists of 7 distinct First-Come-First-Served (FCFS) queues. Each queue serves tasks of a distinct priority level, level 1 being of the highest priority. Operating system activities arrive to the level 1 queue and are not timesliced, but run to completion. Timesharing and batch tasks arrive to the level 2 queue. Levels 2 through 7 are timesliced. The task selection rule is as follows: a task in the level n queue may only go into service if the level 1 through n-1 queues are empty ($2 \leq n \leq 7$).

For purposes of illustration, the workload types in this analysis are restricted to operating system activities, timesharing and batch jobs. The EXEC dispatcher however further classifies operating system activities into several distinct priority types and the user workload may also contain real time, transaction processing and deadline batch programs, all of which run at higher priority than timesharing and batch programs. These additional workload types can straightforwardly be incorporated into the analysis.

Consider a level n task that is currently in service and is receiving a timeslice of length q_n. This task may leave the CPU for one of the following reasons: 1) the task terminates, 2) the task issues a synchronous I/O request. When the I/O is completed, this task will be returned to the end of the level 2 queue. 3) the timeslice expires, in which case the task is returned to the end of the level (n+1) queue, $2 \leq n < 7$. Tasks in the level 7 queue are returned to the end of the level 7 queue. 4) the task is pre-empted by an arrival to the level 1 queue ($2 \leq n \leq 7$) or by an arrival to the level 2 queue ($3 \leq n \leq 7$). The pre-empted task is returned to the head of the level n queue.

A distinctive feature of the EXEC-8 dispatcher is that when pre-empted tasks are re-admitted to the CPU, the task is re-allocated an entire timeslice of length q_n rather than the residual of q_n. This re-allocation of entire timeslices is referred to as timeslice renewal.

Note that the combination of pre-emption, timeslice renewal and class dependent arrival and service rates places the analysis of the EXEC-8 dispatcher beyond the reach of the well-known Feed Back N-level (FB_N) algorithms and their variants [1,3].

3. THE DISPATCHER MODEL

The dispatcher is modelled as a queueing system consisting of a single server, m queues and n customer classes. In our case, m=7 (queue 1 being of highest priority) and n=3, where class 1 represents programs with long CPU processing intervals, class 2 represents programs with short CPU processing intervals and class 3 represents system tasks. Note that class 1 and class 2 might typically represent timesharing and batch programs, though the EXEC dispatcher itself does not make any explicit distinction between these two workload types.

The model assumes that customers arrive to the system according to n independent Poisson processes. Each customer class is further assumed to have an exponential distribution of service times. The routing of the customers is as described in Section 2 and the priority queue is illustrated in Figure 1.

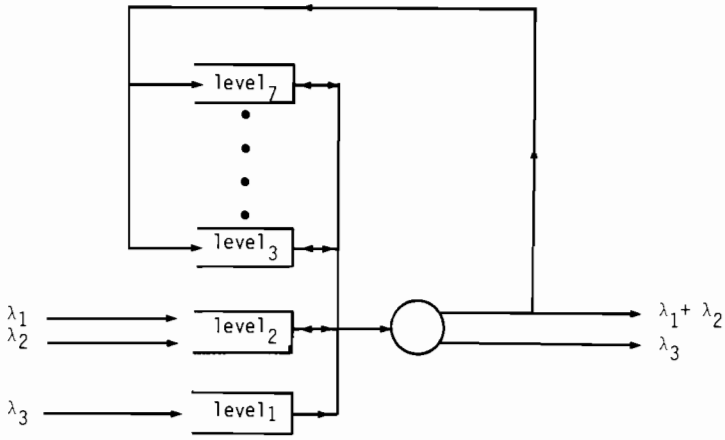


Figure 1. Dispatcher model

3.1 Analysis of the priority queue

A detailed analysis of the priority queueing system illustrated in Figure 1 can be found in van den Heever [4]. We do not intend to repeat the analysis. We shall rather describe the input parameters required by the model, give a brief

summary of the analysis method and then detail why this model is not sufficient for our purposes. The following section 3,2 then outlines how this priority queue can be generalised to give an adequate description of the dispatcher algorithm.

Briefly, the analysis method is based upon the concept of virtual workload V(t) which is defined as the total remaining amount of work (measured in time units) at time t. The distribution of V(t) for an M/G/1 queue is known [5]. It can be shown that the actual amount of work found in the system upon the arrival of a customer of any class is identical to V(t). The analysis then traces the progress of a class j customer as it threads its way according to its priority routing through the several queues illustrated in Figure 1.

The processing of a class j customer is formally described in terms of the following parameters:

λ_j = mean arrival rate of class j customers to the system.

P_j = $(i_1, .. ,i_k, .. ,i_{p_j})$ $\qquad 1 \leq i_k \leq m$

= the route that a class j customer follows through the m priority queues. The integer p_j is the number of passes through the processor required to complete a class j customer. The integer i_k identifies the queue that a class j customer joins on his k th pass through the processor.

e_{jk} = probability that a class j customer completes service and leaves the system on its k th pass through the processor.

λ_{jk} = $\lambda_j \prod_{i=1}^{k-1} (1 - e_{ji})$

= the arrival rate of class j customers to the k th queue.

S_{jk} = a random variable denoting the amount of service that a class j customer receives on its k th pass through the processor.

The analysis proceeds to build a set of linear equations expressing the expected queueing delays experienced by a class j customer on its k th pass through the processor as a function of the variables described above. These equations are then solved, yielding the expected queueing delays, queue lengths and processor utilisations for jobs from each of the customer classes.

However, in order to solve for the expected queueing delays, we must first determine the first and second moments $E(S_{jk})$ and $E(S^2_{jk})$ of the service requests S_{jk} of class j customers on their k th pass through the processor. These moments can easily be calculated once the distribution of S_{jk} is known. The kernel of the matter is that, for the dispatcher algorithm under consideration, these distributions are not straightforward, since they depend not only upon the timeslice lengths and the times between successive I/O operations, but because of the timeslice renewal scheme, they also depend upon the rate of arrival of higher priority jobs. The following section reports how the service time moments can be calculated for timeslice renewal schemes.

3.2 Service time distribution for a customer at level j

Consider a customer of class j on its k th pass through the processor (for notational convenience, the subscripts j and k are omitted from the subsequent analysis). Define the following terms:

- q = preset length of the timeslice for this queue.
- λ = total arrival rate of customers that may pre-empt the customer in service. This arrival process is Poisson.
- ξ = ξ_{jk} = total service time requested by class j customer on pass k.
- Δ = S_{jk} = effective timeslice length as extended by timeslice renewals caused by pre-emptions. Note that only two events can terminate Δ, namely 1) an I/O request 2) a preset timeslice expiry.
- Θ = processing time excluding the final processing interval, where the final processing interval is terminated either by an I/O request or by a timeslice expiry.

Note that $\Delta = \min(\xi, \Theta + q)$

Define:

$$F.(t) = Pr(. \leq t)$$

$$f.(t) = Pr(t \leq . \leq t + \delta t)$$

where $f.(t) = \frac{dF.(t)}{dt}$ if and only if $F.(t)$ is continuous. Define the Laplace-Stieltjes transform $F.(s)$ of $F.(t)$ as

$$F^*.(s) = \int_{-\infty}^{\infty} e^{-st} dF.(t)$$

Note that the general form of the transform has been used where $t \in (-\infty,\infty)$ since we do not preclude $F_.(t)$ having discontinuities at $t = 0$. If $F_.(t)$ is continuous, $F^*_.(s) = f^*_.(s)$ where $f^*_.(s)$ is the Laplace transform of $f_.(t)$

$$F^*_.(s) = \int_0^\infty e^{-st} f_.(t)\, dt$$

The function form of the probability density $f_\Delta(t)$ is straightforward:

$$f_\Delta(t) = \begin{cases} f_\xi(t) & 0 \leq t \leq q \\ \Pr(\xi = t \text{ and the customer acquires} \geq t - q \text{ from timeslice renewal}) & \\ \quad + \Pr(\xi > t \text{ and the customer acquires exactly } t - q \text{ from timeslice renewal}) & t > q \end{cases}$$

..... (1)

The calculation of $f_\Delta(t)$ is simplified by defining an auxiliary variable x_i where x_i denotes the elapsed time between the (i - 1) th and the i th pre-emptions. The roles of x_i, Θ, t and q for the case $t > q$ are illustrated in Figure 2.

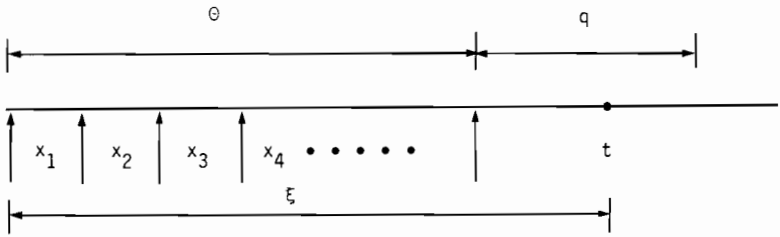


Figure 2a. $\xi = t$ and customer acquires $\Theta \geq t\text{-}q$ from timeslice renewal (customer completes during final processing interval)

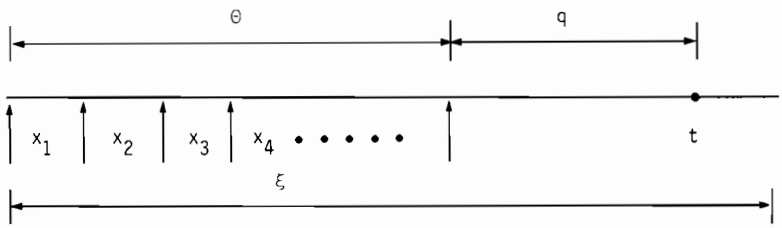


Figure 2b. $\xi > t$ and customer acquires $\Theta = t-q$ from timeslice renewal (customer does not complete during final processing interval)

Equation (1) can be written

$$f_\Delta(t) = \begin{cases} f_\xi(t) & 0 \leq t \leq q \\ f_\xi(t) \int_{t-q}^{\infty} f_\Theta(u)\, du + \{1 - F_\xi(t)\}\, f_\theta(t-q) & t>q \end{cases}$$

and

$$f^*_\Delta(s) = \int_0^\infty e^{-st} f_\Delta(t)\, dt$$

Assuming the service request ξ to be exponentially distributed with mean $1/\mu$

$$f_\xi(t) = \mu e^{-\mu t}$$

then it can be shown

$$f^*_\Delta(s) = (\mu + s)^{-1} \{ \mu + s e^{-(\mu+s)q} F^*_\Theta(\mu+s)\} \quad \text{.....(2)}$$

The next step is to calculate $F^*_\Theta(s)$

$$F_\Theta(t) = \begin{cases} 0 & t < 0 \\ \Pr(\text{no renewal}) & t = 0 \\ \Pr(1 \text{ renewal})\ \Pr(\Theta \leq t \mid 1 \text{ renewal occurs}) & \\ + \Pr(2 \text{ renewals})\ \Pr(\Theta \leq t \mid 2 \text{ renewals occur}) & \\ + \ \ldots\ldots\ldots & t > 0 \end{cases}$$

Where

$$\Pr(\Theta \leq t \mid n \text{ renewals occur}) = \Pr(x_1 + x_2 + .. + x_n \leq t)$$

$$\Pr(\text{timeslice renewed}) = \Pr(1 \text{ or more arrivals during } q) = 1 - e^{-\lambda q}$$

$$\Pr(n \text{ renewals occur}) = \Pr(\text{timeslice renewed})^n \Pr(\text{timeslice not renewed}) = (1 - e^{-\lambda q})^n e^{-\lambda q}$$

Thus

$$F_\Theta(t) = \begin{cases} 0 & t < 0 \\ e^{-\lambda q} + (1 - e^{-\lambda q})e^{-\lambda q} \Pr(x_1 \leq t) & \\ \quad + (1 - e^{-\lambda q})^2 e^{-\lambda q} \Pr(x_1 + x_2 \leq t) + ... & t \geq 0 \end{cases}$$

whereupon it can be shown

$$F^*_\Theta(s) = e^{-\lambda q} / \{1 - (1 - e^{-\lambda q}) F^*_x(s)\} \quad ..... \quad (3)$$

The next step is to calculate $F^*_x(s)$. We make use of the fact that the times between successive arrivals that renew the timeslice are exponentially distributed and cannot exceed q. Therefore

$$F_x(t) = \begin{cases} (1 - e^{-\lambda t})/(1 - e^{-\lambda q}) & 0 \leq t \leq q \\ 1 & t > q \end{cases}$$

whereupon it can be shown that

$$F^*_x(s) = \int_0^\infty e^{-st} \, dF_x(t)$$

$$= \lambda \{1 - e^{-(\lambda+s)q}\}/\{(\lambda+s)(1 - e^{-\lambda q})\} \quad ..... \quad (4)$$

Substituting (3) and (4) into (2):

$$f^*_\Delta(s) = \mu/(\mu + s) + \{s(\lambda + \mu + s)e^{-(\lambda + \mu + s)q}\}/$$

$$\{(\mu + s)(\mu + s + \lambda e^{-(\lambda + \mu + s)q})\}$$

Note that:

(a) $\lim_{q\to 0} f^*_\Delta(s) = 1$. Thus when $q\to 0$, the only value of t that can be realised is t=0, that is the customers can achieve no service at all.

(b) $\lim_{q\to \infty} f^*_\Delta(s) = \mu/(\mu + s)$. Thus, as the timeslice lengths increase, each customer consumes his entire service request (in a single pass) which is exponentially distributed.

(c) $\lim_{\lambda\to 0} f^*_\Delta(s) = (\mu + se^{-(\mu + s)q})/(\mu + s)$. As the arrival rate decreases, timeslice renewal decreases and the queue reduces to a time sliced FCFS exponential server.

(d) $\lim_{\lambda\to \infty} f^*_\Delta(s) = \mu/(\mu + s)$. When timeslices are renewed infinitely fast, each customer is guaranteed to complete and consume his entire service request which is exponentially distributed.

The required moments $E(\Delta)$ and $E(\Delta^2)$ can now be calculated

$$E(\Delta) = -\lim_{s\to 0} \frac{d}{ds} f^*_\Delta(s)$$

$$= 1/\mu - \{(\lambda + \mu)e^{-(\lambda + \mu)q}\}/\{\mu(\mu + \lambda e^{-(\lambda + \mu)q})\} \quad \ldots\ldots\ldots(5)$$

Similarly

$$E(\Delta^2) = \lim_{s\to 0} \frac{d^2}{ds^2} f^*_\Delta(s)$$

$$= 2/\mu^2 - \{2(\lambda + \mu)\, e^{-(\lambda + \mu)q}\, (q + 1/\mu)\}\{\mu(\mu + \lambda\, e^{-(\lambda + \mu)q})\}^{-1}$$

$$+ \frac{2e^{-(\lambda + \mu)q}}{\mu(\mu + \lambda e^{-(\lambda + \mu)q})} \left[1 - \frac{(\lambda + \mu)(1 - \lambda q e^{-(\lambda + \mu)q})}{\mu + \lambda e^{-(\lambda + \mu)q}}\right]$$

It only remains to calculate e_{jk} where

e_{jk} = Pr(class j customer departs from the system after completing service on its k th pass through the processor)

Recall that ξ_{jk} is the total service time requested by a class j customer on its k th pass through the processor. Consider a class j customer who achieves exactly $\Theta + q = y$ seconds on its k th pass. The probability that this customer does not complete, given $\Theta + q = y$, is $\int_y^\infty dF_{\xi_{jk}}(t)$. Therefore

$$\Pr(\text{customer does complete} \mid \Theta + q = y) = 1 - \int_y^\infty dF_{\xi_{jk}}(t) = 1 - e^{-\mu y}$$

Removing the conditioning on $(\Theta + q)$,

$$e_{jk} = \int_0^\infty (1 - e^{-\mu y})\, dF_{\Theta+q}(y)$$

$$= 1 - \int_0^\infty e^{-\mu y}\, dF_{\Theta+q}(y)$$

$$= 1 - F^*_{\Theta+q}(\mu)$$

$$= 1 - F^*_{\Theta}(\mu-q) = 1 - e^{-\mu q} F^*_{\Theta}(\mu)$$

From equations (3) and (5), the above reduces to

$$e_{jk} = \mu\, E(\Delta_{jk})$$

4. DISPATCHER PERFORMANCE PREDICTION

Figures 3 through 7 summarise the model's predictions of dispatcher performance as a function of the total arrival rate Λ where $\Lambda = \lambda_1 + \lambda_2 + \lambda_3$. The mean service requests of the various job classes were chosen to be $1/\mu_1$ = 7 msecs for short jobs, $1/\mu_2$ = 60 msecs for long jobs and $1/\mu_3$ = 700 microsecs for system tasks. Workload composition was maintained by keeping the ratio $\lambda_1 : \lambda_2 : \lambda_3$ constant.

One measure of dispatcher performance is its response time, defined as the average time elapsed between a job joining the dispatcher queue and leaving the dispatcher, either to do synchronous I/O or because it has terminated. Figure 3 illustrates the dispatcher response time as a function of the total arrival rate Λ. Notice that short jobs maintain a better response at higher pre-emption

rates than longer jobs, thus complying to the fundamental requirement of any dispatcher algorithm. The response of long jobs on the other hand rapidly degrades with increasing Λ.

The main purpose of the dispatcher model is to continually furnish the R-MPA control system with optimal values for the preset timeslice lengths. Figure 4 reveals the model's ability to identify the preset value of the first timeslice which minimises response for short jobs. Note that the optimum value of this timeslice does not depend upon the arrival rates and service requirements of the long jobs present in the workload.

Figure 5 illustrates the extent to which timeslice lengths are extended by renewals; preset values for these results were q_2 = 2 msecs and q_3 = 16 msecs. Due to the extension of timeslice lengths, few jobs enter the lower priority queues 4 through 7 until high arrival rates saturate the system. Figure 6 confirms that for $\Lambda < 70$ arrivals per second, about 80 percent of the jobs are dispatched from queues 2 and 3.

Figure 7 reveals the distribution of queue lengths for short and long jobs for two different arrival rates, $\Lambda = 30\ \text{sec}^{-1}$ and $\Lambda = 90\ \text{sec}^{-1}$. (Pre-empted jobs which have already received some service are not considered to be in the waiting line). Note that the results presented in figure 7 cast some doubt on the wisdom of the timeslice renewal policy. The goal of this policy is to prevent long jobs from migrating to the very low priority queues where they will remain without service while jobs on higher levels are being served. This goal is indeed achieved. However, this positive effect is counterbalanced by the fact that, under heavy load conditions, all jobs receive longer timeslices. Therefore, under heavy load conditions, the order of arrival of tasks to the dispatcher becomes an important factor in determining dispatcher response.

5. ACKNOWLEDGEMENT

The authors are indebted to their colleague, Roelf van den Heever, whose ideas greatly contributed to the solution of the problem.

REFERENCES:

1. Coffman, E.G. and Kleinrock, L.: Feedback queueing models for time-shared systems. JACM Vol. 15 No. 4 (Oct. 1968) pp 549 - 576.

2. Kritzinger, P., Krzesinski, A. and Teunissen, P.: Design of a control system for a timesharing computer system. Proc. Int. Conf. on Performance of Computing Installations. ICPCI 78, Lake Garda, Italy, pp 103 - 114 (June, 1978). North Holland: Amsterdam.

3. Schrage, L.E.: The queue M/G/1 with feedback to lower priority queues. Management Science, Vol. 13 No. 7 (March, 1967) pp 466 - 474.

4. Van den Heever, R.J.: A planning tool for computer priority scheduling. Research Report RC 6238. IBM T.J. Watson Research Centre, Yorktown Heights, New York (October, 1976).

5. Wolff, R.W.: Time sharing with priorities. SIAM J. Appl. Math. Vol. 19 No. 3 (November, 1970) pp 566 - 574.

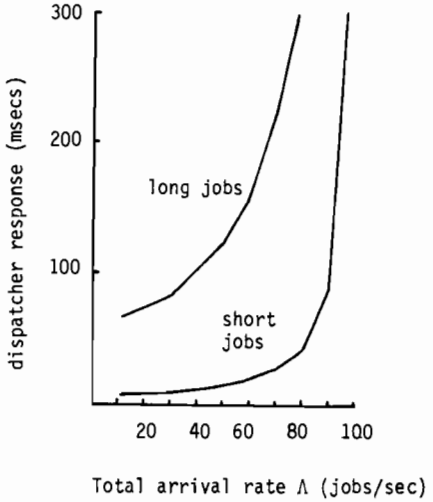


Figure 3

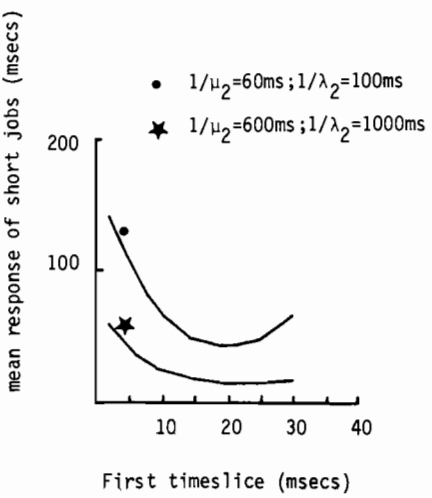


Figure 4

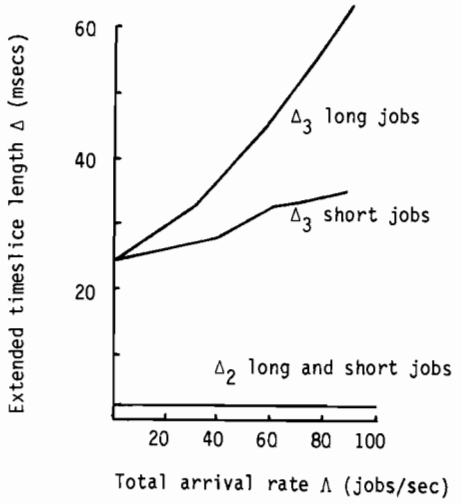


Figure 5

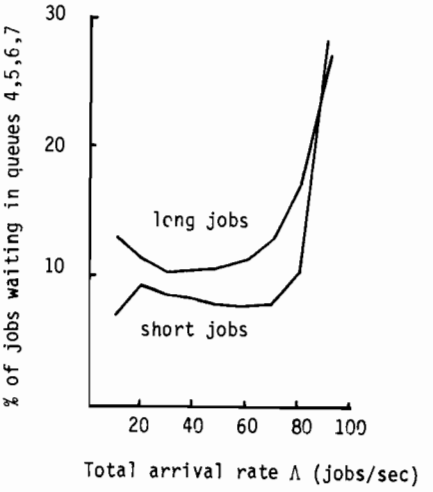


Figure 6

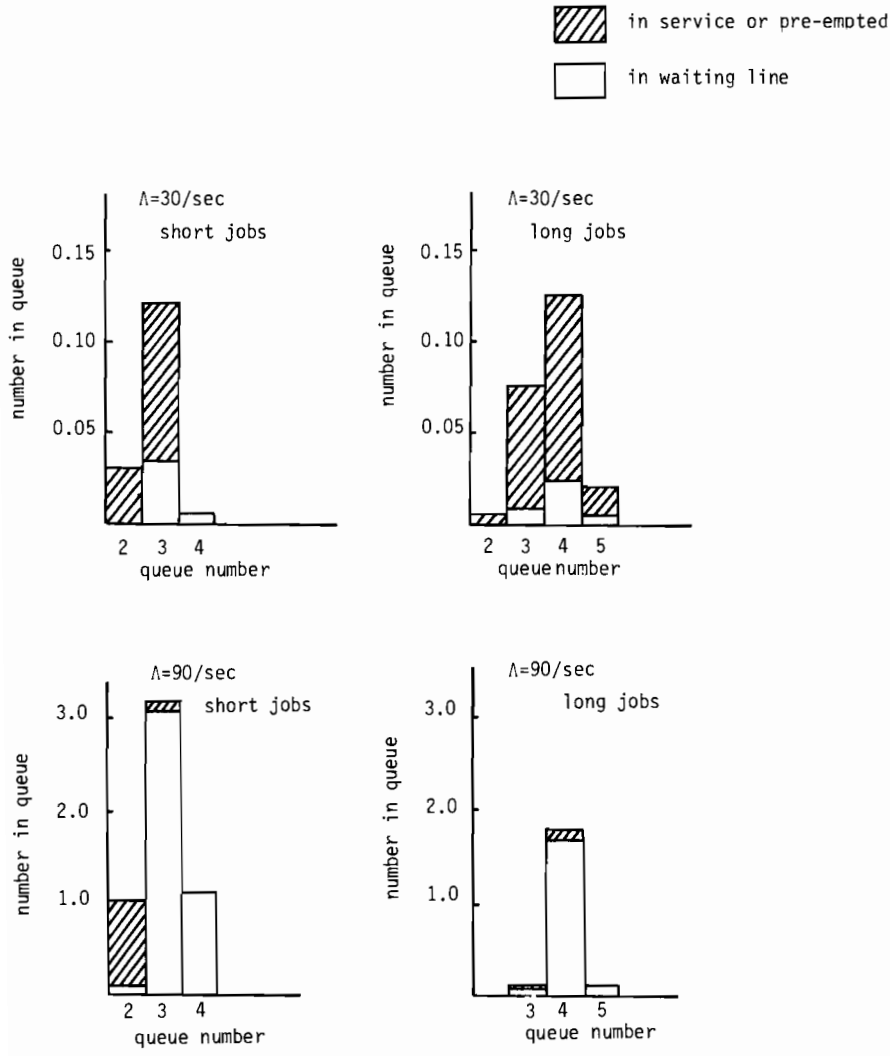


Figure 7. Queue length distribution

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

A STUDY OF A MECHANISM FOR CONTROLLING MULTIPROGRAMMED MEMORY IN AN INTERACTIVE SYSTEM

A. Brandwajn (+,++) and J.A. Hernandez (++)

\+ Duke University, Durham, N.C. 27706, USA
++ Ecole Nationale Supérieure des Télécommunications
46, rue Barrault 75634 Paris, France

This paper deals with the following mechanism for controlling the multiprogramming set in a demand paging system: processes are dynamically divided into several categories according to the number of page faults generated during their residence in main memory. A process is admitted into the multiprogramming set only if there is enough space free in the main memory to contain the number of pages corresponding to the current category of the process. Using a queueing network model of an interactive system with such a control mechanism we study the effectiveness of the control considered, and, more particularly, its ability to partition the memory space according to the locality of processes.

1. Introduction

Since the experimental studies of the dynamic program behaviour (1), and the introduction of the notion of locality and of the working set model (2), a number of system designers have attempted to include these results in their memory management algorithms (e.g. (3)). An interesting attempt to dynamically partition the memory space according to the observed behaviour of processes was implemented in the interactive virtual memory system ESOPE (4).

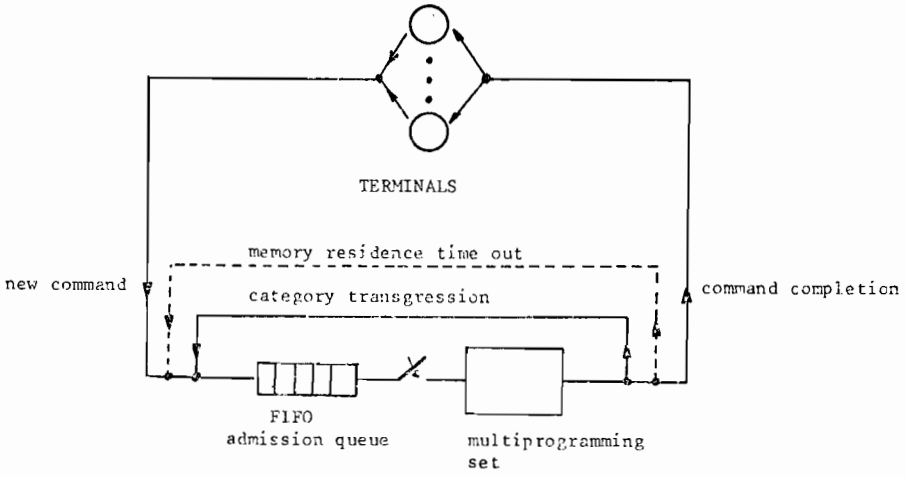


Figure 1: framework for algorithm considered

The algorithm, apparently inspired from the EMAS system (5), is as follows (see Fig. 1): processes are divided into categories, and a process is admitted into the multiprogramming set (i.e. allowed to share real memory and to compete for other system resources) only if there is enough space free in main memory to contain the number of pages corresponding to the current category of the process. This number of pages are reserved for the process, but the actual fetch takes place on a page on-demand basis. The process category is adjusted, basically, in two instances:

i) upon command completion: the category is reduced by one (if possible), if the number of pages fetched (i.e. of pages faults generated) is smaller than the corresponding limit for the immediately preceding (i.e. lower) category;

ii) upon category transgression: if a process attempts to access more pages than reserved for it according to its current category, the process is ejected from the multiprogramming set. The memory pages that have been allocated to it are freed, and its category is increased by one (if possible). The process is then placed at the end of the admission queue.

A new process entering the system is assigned a category on an arbitrary basis.

The intuitive motivation behind this algorithm seems to be the following. The current category of a process is used as an estimate of its working set size. In order to prevent thrashing (6), processes are admitted into main memory only if the latter can contain their estimated "working sets". Note that this algorithm not only controls the partitioning of the memory space (and thus the multiprogramming level) but also automatically ensures replacement of pages (when a process leaves the multiprogramming set). Note also that it reacts to instantaneous changes in program behaviour, and thus introduces a strong coupling between the system execution and control functions. This hinders the system's decomposability (7).

In summary, the algorithm controls the multiprogramming set via dynamic memory partitioning. The latter depends on an on-line process classification based on the virtual time paging behaviour of a process. The number and limits of the categories are the parameters of the algorithm. Therefore, it is important to study the system throughput (or, equivalently, the mean response time) as a function of the category assignment, and its dependence on system and program behaviour parameters. It is also important to study the effectiveness of the virtual time on-line classification of processes. (The latter point is suggested by the obvious result of recent modelling studies (see e.g. (8)) which indicates that the number of pages needed by a process "to be executed efficiently" (9) is not an absolute process characteristic but depends strongly on e.g. the average service time of the secondary memory device). These questions are addressed in this paper.

The problem of multiprogrammed memory management, and, more particularly, that of controlling the level of multiprogramming has received a considerable attention in the litterature, e.g.(10-19) are but a few recent references. The work by Schoute (17) seems the closest to this paper. Our work differs from (17) in that we explicitly take into account the effect of program loading (20, 21) on the performance of the admission mechanism, i.e. on the current category of processes. This allows us to study the influence of program behaviour on system performance, and, in particular to consider the case where the processes belong to different classes as regards their locality and total compute time.

Let us now define the scope of our paper. We shall restrict our attention to a situation where only interactive processes are present in the system (no background batch jobs) and where the number of logged-in (active) terminals is constant over periods of time long enough for a stationary analysis to be valid. The arbitrary initial category assignment will thus be neglected and only the long-run behaviour of the algorithm under consideration will be studied. It may be noted that in the algorithm as implemented in the ESOPE system, processes are

ejected from the multiprogramming set if their residence time in real memory exceeds a given limit. We concentrate on the paging behaviour of processes and, therefore, neglect this mechanism.

To start with, we shall assume that all the users are statistically identical and independent. Their paging behaviour will be modelled by the life-time function (1) which relates the average process execution time between two successive page faults, e(m), to the amount of memory m, allocated to the process. We shall use the two-parameters fit proposed in (9)

$$e(m) = 2b/(1+(d/m)^2), \qquad (1.1)$$

which accounts for the saturation effect at larger m. Figure 2 shows examples of the life-time function for several sets of parameters b and d.

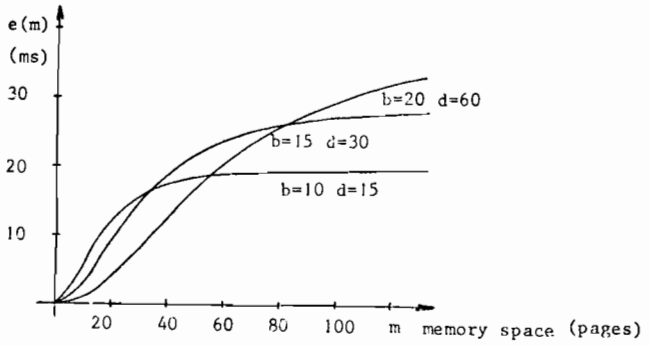


Figure 2: examples of life-time curves

I/O activity other than caused by paging, and most overheads will be neglected.

The next section is devoted to the description of a queueing model of the system under study. In Section 3 the numerical results obtained with a single class of processes are discussed. Section 4 is devoted to the numerical results with several classes of processes.

2. A queueing network model

The queueing network model we shall use to study the properties of the algorithm under consideration is represented in Figure 3. The behaviour of a user is modelled by a sequence of think times followed by a generation of a command after which the user remains inactive until the system response (22). (A somewhat more elaborate model of user interaction has been proposed recently in (23)). The user think time is assumed to be an exponentially distributed random variable with mean $1/\lambda$; the set of terminals is thus represented by an exponential server with service rate $n_c\lambda$, n_c being the current number of active terminals. We assume that there are q categories in the system, and we denote by m_j the page number limit for category j, j= 1,...,o, where $m_1 < m_2 < \ldots m_{q-1} < m_q$. A command generated by a user (a process) enters a FIFO admission queue (AQ). According to the admission mechanism described, if the process at the front of AQ is of category j, it will be admitted only if there are at least m_j memory pages available

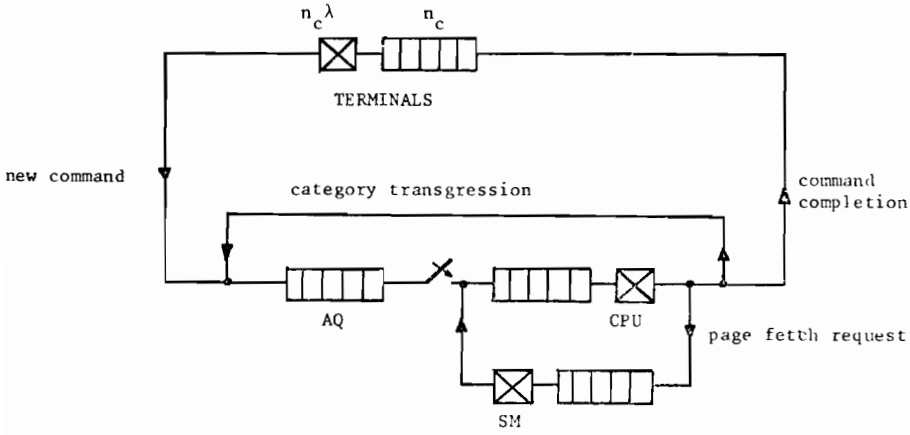


Figure 3: a queueing network model

The multiprogramming set is represented by a CPU and a secondary memory (SM) device with their queues of processes. We denote by $v_1(m)$ the page fault rate of a process which has m pages present in real memory, and we let

$$v_1(m) = \begin{cases} 1/e(m) & , m=1,\ldots,m_j \ ; \\ 1/\omega & , m=0 \ ; \end{cases} \qquad (2.1)$$

where e(m) is given by (1.1), and ω is the average system overhead time per process admission into the multiprogramming set. A process having requested a page is either placed in the SM queue if $m< m_j$, or ejected if $m=m_j$. The mean service time of the SM is $1/u_1$, and it is assumed to account for a possible preliminary saving of a written-onto memory page before the actual fetch of the page requested. We assume that the SM service time is an exponentially distributed random variable, and that the SM queueing discipline is FCFS.

In the case of ejection, the memory pages belonging to the process are freed, and the process category is adjusted (j:=j+1, if j< q). The process is then placed in AQ, and, once admitted, will have to reacquire one by one its pages.

The service discipline at the CPU is assumed to be Processor Sharing. The CPU service time for a process is assumed to be an exponentially distributed random variable with rate

$$v(m) = \begin{cases} v_1(0) & , \text{ if } m=0 \\ v_1(m)+v_0 & , m=1,\ldots,m_j, \end{cases} \qquad (2.2)$$

for a process of category j with m pages in memory. v_0 denotes the rate of command completions. We let

$$v_0 = 1/c \qquad (2.3)$$

where c is the mean total CPU time per command.

If a process of category j has no more than m^*_{j-1} pages in memory at the moment of completion its category is adjusted (j:=j-1, if j>1).

At this point, we shall use the mean system response time, W, as a measure of overall system performance and of the utilization of the CPU system resource (using the Little's formula (24), one can easily show that $W=cN/B - 1/\lambda$, where N is the total number of terminals and B denotes user mode CPU utilization).

Note that despite a number of assumptions aimed at the tractability of the model its solution is far from obvious. A direct brute-force attack of the model balance equations would be difficult in practice because a very detailed state description is needed. The latter seems particularly inadequate with respect to the performance measures defined.

We summarize below the four main steps of the approximate hybrid solution method we have used. The interested reader may refer to (32) for details.

Step 1 We analyze the behaviour of a user in its virtual (i.e. execution) time so as to obtain, for an active process of category j, j=1,...,q:

- f_j, the rate of page faults;
- w_j, the rate of category transgressions;
- z_j, the rate of command completions;
- s_j, the probability that process category will decrease upon command completion. This is similar to the approach used in (20, 21), and yields a simple analytical solution.

Step 2 We use the average page fault rates f_j in a simple model of the multiprogramming set (see Figure 4) to compute $A_j(\underline{n})$, the CPU utilizations for processes of each category under a constant load of $\underline{n}=(n_1,\ldots, n_q)$, where n_j is the number of category j processes in the multiprogramming set.

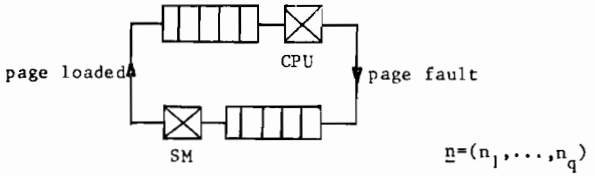


Figure 4: multiprogramming set submodel

Due to the assumptions on service time distribution and on queueing disciplines, the analytical solution of this model may be easily obtained (25).

Step 3 We use the $A_j(\underline{n})$, w_j, z_j and s_j in a queueing model of the multiprogramming set control. The model (see Figure 5) reflects only events pertaining to the operation of the admission control, and is studied under constant load of ℓ processes in order to obtain $u(\ell)$, the rate of command completions with a total of ℓ processes (waiting and admitted).

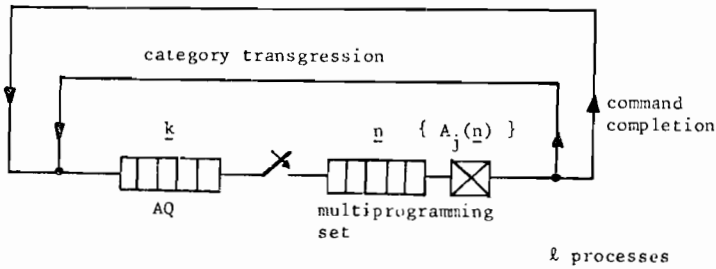


Figure 5: multiprogramming control model

The multiprogramming set is represented by an exponential queue. Its current state is described by $\underline{n}=(n_1,\ldots, n_q)$, the vector of the numbers of processes of each category currently in the multiprogramming set. The rates of departures from the latter are set to be the products of $A_j(\underline{n})$ by the corresponding rate obtained in Step 1, e.g. the rate of category transgressions for processes of category j is $A_j(\underline{n})\ w_j$. The state of the admission queue is described by $\underline{k}$, the vector of process categories in FIFO order.

The model can be solved using discrete event simulation (26). A confidence interval for the result may be derived applying the regenerative simulation method (27), so that we obtain two numbers $u'(\ell)$ and $u''(\ell)$ which are the bounds of the confidence interval for the rate of command completions with a total of ℓ processes.

Step 4 We analyze the system behaviour at a highly aggregate level via the queueing model of Figure 6.

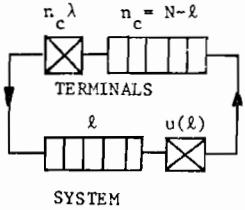


Figure 6: aggregate system model

The terminals and the system are represented by exponential servers with service rates $n_c\lambda$ and $u(\ell)$, respectively ($n_c+\ell=N$, the total number of terminals). The analytical solution of this model is well known and allows an easy computation of the average number of processes in the system, and, hence, via the Little's formula (24), of the mean system response time. In our case, since $u(\ell)$ is given in the form of an interval, interval arithmetic has to be used to finally obtain W' and W", the two bounds of the confidence interval for the mean system response time. Note that we have used a decomposition technique whereby one computes approximate values for conditional probabilities by analyzing subnetworks under constant (full) load conditions (28, 29). As a whole, our approach is similar to that discussed in (30), in that at each step in the solution of the decomposed problem, the solution method which seems the most appropriate (analytical, discrete-event simulation,...) is used.
The next section is devoted to numerical results obtained from our model.

3. Numerical results with a single class of processes.

Since the system under consideration is quite complex, and both the model elaborated and its solution involve a number of simplifications and approximations, it is important to gain some confidence in the accuracy of the results before using them to draw conclusions. Therefore, we have tested our model using results of measurements of the ESOPE system under simulated load (31). A few modifications had to be introduced in our model to reflect the actual operation of the ESOPE system, and to be able to use measurement results. Mainly, in ESOPE not all page faults result in a page fetch. Some of the pages deallocated at the moment of process ejection, and requested again during subsequent residence periods, may still be present in main memory. Such pages are simply "recovered" by table update. The number of pages actually fetched has been measured. The ratio of the latter number to the total number of pages faults will be denoted by $1-\beta$, so that β is the relative frequency of pages recoveries. Hence, the actual page fetch rate becomes $f_j' = (1-\beta)\ f_j$, and this value has to be used in Step 2 to compute the CPU utilization.
We also let

$$v(m)=\begin{cases}\text{"}\infty\text{"} & , m=0,\\ v_1+v_0 & , m=1,\ldots,m_j,\end{cases} \qquad (4.1)$$

where v_1 is the average page fault rate (i.e. total number of page faults divided by total CPU time), v_0 is the rate of command completions (i.e. total number of interactions divided by total CPU time).
Table 1 shows the results of model tests. A 90% confidence interval will be used throughout this paper.

Table 1

values of model parameters:
category limits in number of pages (q=10):
m_1=4; m_2=8; m_3=16; m_4=24; m_5=32; m_6=48;
m_7=64; m_8=80; m_9=112; m_{10}=148;
category reduction:
$m_j^*=m_{j-1}-1$, j=2, 3,..., 10;
total memory available for paging: M=184 pages;

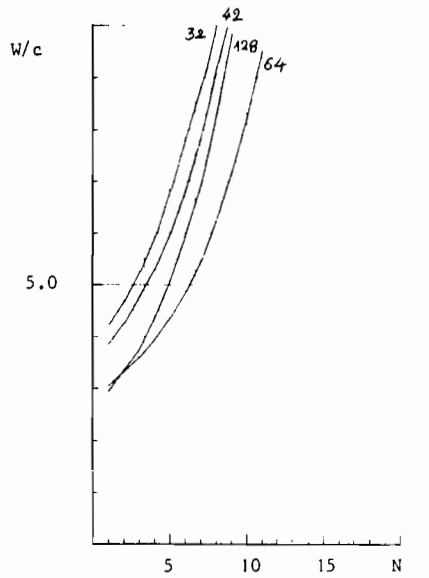
I N=5 (number of active terminals)
v_0 =1/c=1/464 ms ; v_1 =0.04 ms ; β=0.76;
$1/u_1$ =37 ms (mean service time per page fetch);
$1/\lambda$ =10.4 s (mean user think time);
response time W measured : 0.70 s
obtained from model : (0.77 s , 0.85 s)
II N=5
v_0 =1/458 ms ; v_1 =0.058 ms ; β=0.69; $1/u_1$ =38 ms ; $1/\lambda$ =10.8 s;
response time W measured : 0.89 s
obtained from model : (0.93 s , 1.04 s)
III N=12
v_0 =1/490 ms ; v_1 =0.038 ms ; β=0.47; $1/u_1$ =41.1 ms; $1/\lambda$ =11.8 s;
response time W measured : 1.44 s
obtained from model : (1.37 s , 1.56 s)

Clearly, the model may be used with reasonable confidence.

We now first use it to study the influence of system and program behaviour parameters on the choice of category limits. To start with, we consider the case where the number of categories is equal to one (q=1). The category mechanism then results in a fixed maximum degree of multiprogramming. For a given main memory size (available for paging), only a few discrete values of category limit m_1 have to be studied, viz. those for which m_1 is maximum with a given resulting degree of multiprogramming. All other values of m_1 may only cause the page fault rate f_1 to increase with no compensation by the multiprogramming effect, since the multiprogramming degree will not increase. Thus for a memory of M=128 pages, the values of m_1 to consider are 128, 64, 42 pages, etc. Note that with one category the solution Step 3 may be easily carried out analytically.

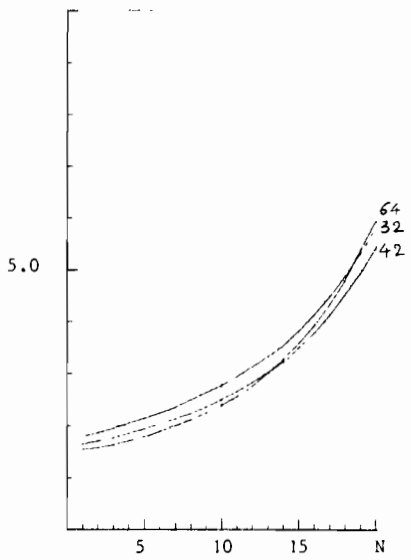
We have represented in Figure 7 the average relative system response time i.e. the ratio W/c, versus the number of terminals, N, for a set of model parameters with the mean service time of the secondary memory device, $1/u_1$, set to 20 ms. Figure 8 shows the results obtained with $1/u_1$ set to 5 ms, the other parameters remaining unchanged. We observe the important effect of the mean service time of the secondary memory device on system performance, and, in particular on optimum (i.e. which minimizes mean system response time) category assignment. The latter changes from 64 pages for the set of parameters used in Figure 7, to 42 pages in Figure 8.

(In Figures 7 through 14 the following values of model parameters are used: M=128 pages, $1/\omega$=0.01 ms).



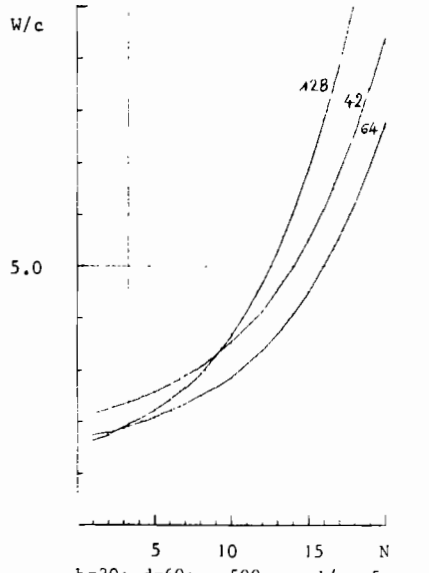
b=15; d=30; c=500 ms; $1/u_1$=20 ms;

Figure 7



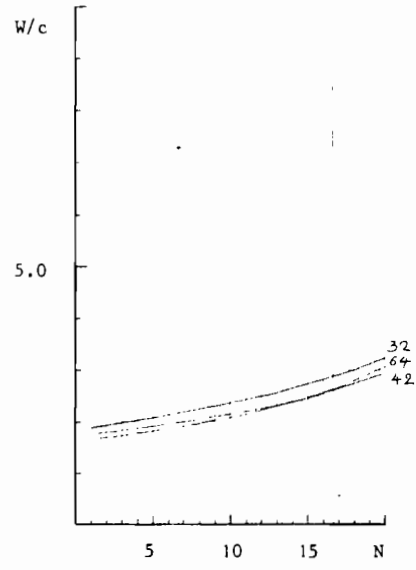
b=15; d=30; c=500 ms; $1/u_1$=5 ms;

Figure 8



b=20; d=60; c=500 ms; $1/u_1$=5 ms;

Figure 9



b=15; d=30; c=250 ms; $1/u_1$=5 ms;

Figure 10

The effect of program behaviour may be seen in Figures 8 and 9 in which the values of life-time function parameters used are b=15 ms, d=30 pages, and b=20 ms, d=60 pages, respectively. Again, we observe that the optimum category assignment changes. Similar observation may be made as regards the influence of the average total CPU time per command, c, in Figures 8 and 10, c being set to 500 ms and 250 ms respectively. As a whole, the figures obtained clearly indicate that the optimum limit assignment for one category may be sensitive to both system and program behaviour parameters. In particular, an important and undesirable effect to be noted is that the optimum category assignment may depend on the number of logged-on terminals. This may be seen in Figure 8 where the category limit of 64 pages yields better results than the 42 pages limit up to N=12 terminals. The curves of Figure 10 exhibit a similar effect.

As a next step in the study of the category mechanism we consider the case where the number of categories is sufficiently large so that two successive category limits differ only by a small number of pages. E.g. for a total memory space of 128 pages and q=32 with equidistant category limits, two adjacent categories differ only by 4 pages. A high number of categories results in excessive computational difficulties, especially in Step 2 of our solution procedure. It may be noted, however, that the probability distribution of the current category of an active process (denoted p(j)) is highly non uniform. As a consequence, if we use only the few most probable categories in Steps 2 and 3, the error introduced in the final result should not be important (this reduction of the number of categories has been actually used in the model validation for N=12 terminals). Figure 11 shows a few examples of the above probability distribution. We have represented there p(j) versus j, the category number, in the case of equidistant categories with q=16 and 32, for two different life-time functions. We observe that p(j) is non negligible only for some five to six values of j out of 16 or 32.

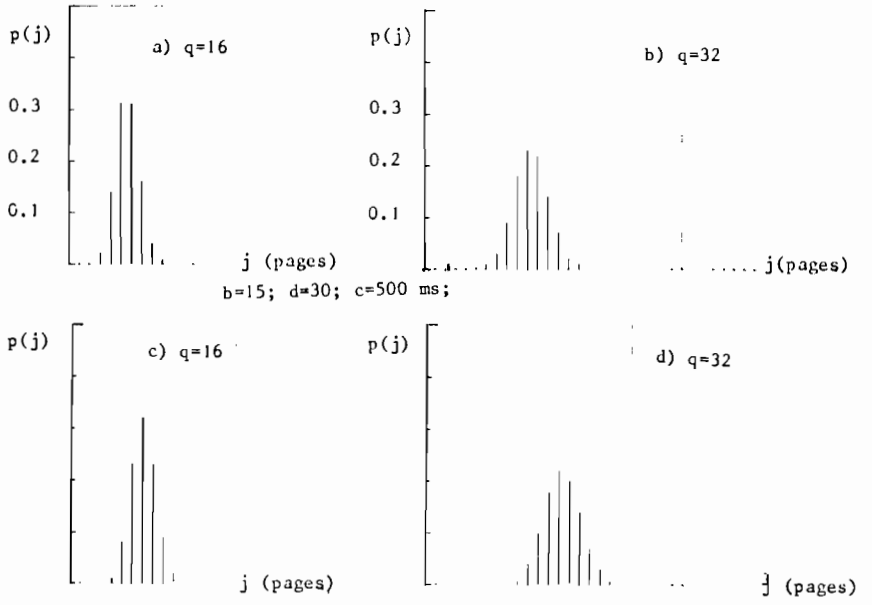
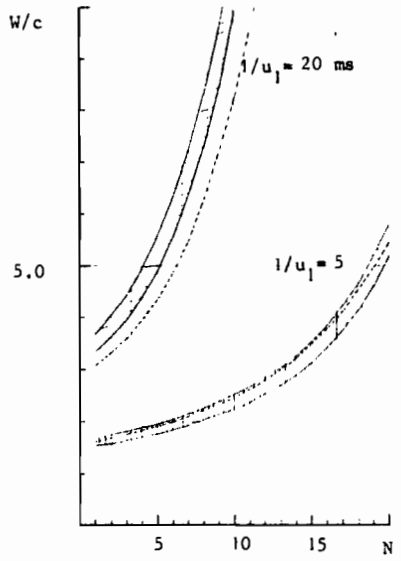


Figure 11

In Figure 12 we have plotted the results obtained (using the five most probable equidistant categories out of 32) for the values of model parameters used in Figures 7 and 8.



b=15; d=30; c=500 ms;

Figure 12

A comparison of the curves indicates that a large number of categories, in the case studied, does not improve the system response time. The effect may even be adverse, as illustrated by the curves for $1/u_1$=20 ms. It is not difficult to understand why this is so. While with only one category the latter may be adjusted to match best the speed of the paging device, in the case of a large number of categories there is not much adjustment possible. The category mechanism "classifies" the processes according to their virtual time page fault rates. There is, in general, no reason for this classification to correspond to the optimum multiprogramming level, since the latter may be very sensitive to the speed of the paging device. So far we have considered only a single class of processes. It is interesting and important to determine whether our observations carry over to the case where the processes form several classes as regards their paging characteristics and total CPU requirements. The next section is devoted to this subject.

4. Several classes of processes

We now aasume that there are ξ classes of processes present in the system. We denote by c_i, ($i=1,\ldots,\xi$) the average total CPU time for a class i command, and by $e_i(m)$ the mean CPU time between successive page faults for a class i process with m pages in main memory. Using obvious notational generalizations we have

$$e_i(m)=2b_i/(1+(d_i/m)^2)\ , \tag{5.1}$$

$$v_{ii}(m)=\begin{cases}1/e_i(m)\ , & m=1,\ldots,m_j\\ 1/\omega\ , & m=0,\end{cases} \tag{5.2}$$

$$v_{oi}=1/c_i\ , \tag{5.3}$$

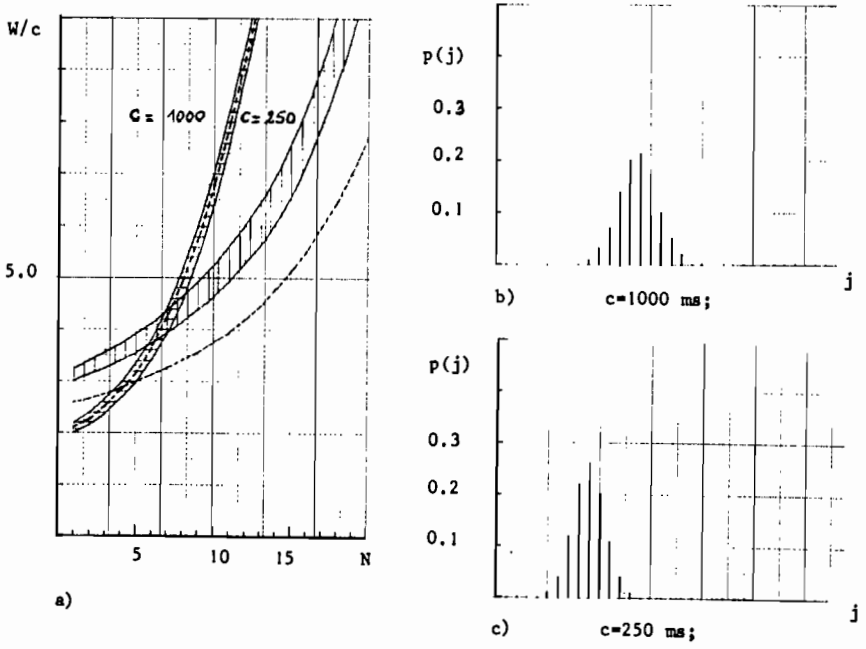
$$v_i(m) = \begin{cases} v_{1i}(0) & \text{,if } m=0 \text{ ,} \\ v_{1i}(m)+v_{oi} & \text{,if } m=1,\ldots m_j \text{ ,} \end{cases} \qquad (5.4)$$

for $i=1,\ldots, \xi$.

We denote by p_i, ($\sum_{i=1}^{\xi} p_i = 1$) the probability that a newly generated command is of class i.

The analysis presented in Section 2 may be easily extended to include several classes of processes. In fact, it suffices to modify the solution Step 1 (i.e. the analysis of process behaviour in its virtual time), the other steps remaining unchanged. The detail of the modification is given in (32). It is worth noting that we obtain a well-decomposed, easy to evaluate recurrent solution for Step 1. Hence, our model has no difficulty in coping with many classes of processes.

We now discuss the numerical results obtained from our model with classes of processes. We have studied the case when there are two classes of processes, equally probable ($p_1=p_2=0.5$), with different memory locality as represented by the life-time function parameters b_i and d_i. In Figure 13a we have plotted the relative average response time versus the number of terminals for a set of model parameters with 32 equidistant categories and two different values of the mean total compute time per command: $c_i=1000$ ms, i=1,2, and $c_i=250$ ms, i=1,2. The dotted curves correspond to the optimum category assignment with only one category.



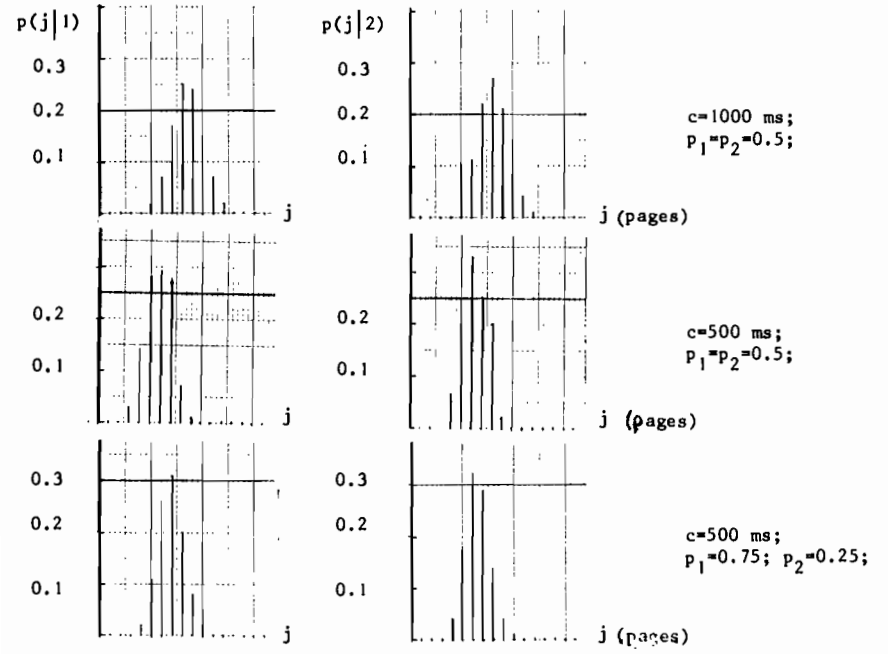
$b_1=15$; $d_1=30$; $b_2=20$; $d_2=60$;
$p_1=p_2=0.5$;

$1/u_1=10$ ms; $q=32$; $c_1=c_2=c$;

Figure 13

We observe that not only a large number of categories does not result in an improvement over the optimum with one category, but may yield significantly worse results. This is in particular the case for c_i=250 ms, i=1,2. We hence conclude that an effective control of the multiprogramming set cannot be based only on the virtual time behaviour of processes, like the "classification" performed by the category mechanisms. Parameters such as e.g. the speed of the paging device must be included in the control mechanism. We have represented in Fig. 13 b and c the virtual time probability that a process belongs to category j, p(j), for c=1000 ms and c=250 ms, respectively. We observe the important effect of the average total compute time on the process classification.

Since the category mechanism with many categories fails to control efficiently the multiprogramming set, and since its primary motivation is to provide estimates of the "working-set" sizes for different processes, i.e. to classify processes according to their locality, it is interesting to study its performance with respect to this objective. We have therefore computed the stationary (virtual time) conditional probability that a process is classed in category j (j=1,...,q) given that it is of class i (i=1,...,ξ), p(j|i). The results obtained are represented in Figure 14. They clearly indicate that this probability strongly depends on the total compute times c_i (which we have already observed), and on the relative frequencies of different classes of commands, p_i. In particular, increasing the total compute time, c_i, results in a classification in a higher category. A possible explanation for this effect is that with larger c_i more pages are referenced, and the category mechanism is such that when new pages are to be loaded the category must, quite often, be increased. Therefore, the most probable category for a given class of processes does not, in general, correspond to a "good" point on the life-time curve of the process.



b_1=20; d_1=60; b_2=15; d_2=30; q=16;

Figure 14

5. Conclusion

We have presented a study of the category mechanism for multiprogrammed memory management. This study has been performed in the context of a demand-paged interactive system. A queueing network model of such a system has been built, and a hybrid (analytical-discrete event simulation) decomposed solution has been obtained for classes of processes. The model neglects I/O activity other than paging, and overheads other than that incurred when a process is admitted into the multiprogramming set. (Numerical results, not reported in this paper, show that the latter overhead has little influence on the performance of the admission mechanism). These features can be relatively easily incorporated in our model, and require that only Step 2 (i.e. the analysis of the CPU utilizations) of our solution procedure be modified.

The numerical results obtained indicate that the system performance (as measured by the relative average response time) with several categories is, in general, worse than the optimum performance with a single category. This seems to be true both for cases when there is a single class and when there are several classes of processes with distinct paging behaviour characteristics. Moreover, the category which is the most probable in the virtual time of a process of a given class does not seem to reflect in any clear way the paging characteristics of the process, as represented by its life-time curve.

Hence, we must conclude that -for the type of life-time curves considered- the category mechanism fails both in optimizing the level of multiprogramming and in classifying the processes according to their paging characteristics i.e. in dynamically partitioning real memory. We interpret the former failure as indicating that an effective control of multiprogramming, in general, cannot be based solely on virtual time behaviour of processes. Although algorithms including the speed of the paging device have been proposed (e.g.(11)), the problem of optimal multiprogramming seems to require further study. Indeed, a (near) optimal policy should also incorporate features such as various I/O's and overheads (including that of the policy itself) since these may be shown to importantly affect the optimal multiprogramming level.

An interesting point for further investigation is to determine to what extent our findings could be affected by the shape of the life-time curve. In particular, it will be important to study the multicategory versus the monocategory performance in the case where the life-time curves of processes possess distinct sharp "knees".

References

1. Belady,L., Kuehner,C.J. : Dynamic space sharing in computer systems. Comm. ACM 12, 282-288 (1969).
2. Denning,P.J. : The working set model for program behavior. Comm. ACM 11, 323-333 (1968).
3. Morris,J.B. : Demand paging through utilization of working sets on the MANIAC II. Comm. ACM 15, 867-872 (1972).
4. Bétourné,C.,Kaiser,C.,Krakowiak,S.,Mossiere,J. :Process management and resource sharing in the multiaccess system ESOPE. Comm. ACM 13,727-733 (1970)
5. Whitfield,H., Wight,A.S. : EMAS - the Edinburgh Multi-Acces System. Computer Journal 16, 331-346 (1973).
6. Denning,P.J. : Trashing : its causes and prevention. In AFIPS Conference Proceedings, FJCC 33.
7. Courtois,P. : Decomposability, instabilities and saturation in multiprogramming systems. Comm. ACM 18, 371-377 (1975)
8. Brandwajn,A., Buzen,J., Gelenbe,E., Potier,D. : A model of performance for virtual memory systems. Proc. ACM SIGMETRICS Symp. 9 (October 1974).
9. Chamberlin,D., Fuller,S., Liu,L. : An analysis of page allocation strategies for multiprogramming systems with virtual memory. IBM J. Res. and Develop. 17 404-412 (1973).

10. Brandwajn,A. : A model of a time-sharing system with two classes of processes. Gesellschaft fuer Informatik 5 Jahrestagung, Dortmund, Springer Verlag, 547-566 (Oct. 1975).
11. Denning,P.J., Kahn,K.C., Leroudier,J., Potier,D. : Optimal multiprogramming. Acta Informatica 7, 197-216 (1976).
12. Neilson,J.E. : An analytic model of a multiprogrammed batch time -shared computer. In Proc. Int. Symp. on Computer Performance, Measurement and Evaluation, Harvard Univ., Cambridge,Mas., 59-70 (March 1976).
13. Landwehr,C.E. : An endogenous priority model for load control in combined batch-interactive computer systems. In Proc. Int. Symp. on Comp. Perf. Meas. and Eval., Harvard Univ., Cambridge, Mas., 282-295,(March 1976).
14. Reiser,M., Konheim,A.G. : Queuing model of a multiprogrammed computer system with a jobqueue and a fixed number of initiators. In Modelling and Performance Evaluation of Computer Systems, Ispra, Italy, 319-334 (Oct. 1976).
15. Hine,J.M., Mitrani,I., Tsur,S. : The use of memory allocation to control response times in paged computer systems with different job classes. In Model. and Perf. Eval. of Comp. Syst. Ispra, Italy 201-216 (Oct. 1976).
16. Coffman,E.G., Ryan,T.A. : A study of storage partitioning using a mathematical model of locality. Comm. ACM 15, 185-190 (1972).
17. Schoute,A.L. : Comparison of global memory management strategies in virtual memory systems with two classes of processes. In Model. and Perf. Eval. of Comp. Syst., Ispra, Italy, 389-414 (Oct. 1976).
18. Bard,Y. : The modeling of some scheduling strategies for an interactive system. In Int. Symp. on Computer Perf. Model., Measurement, and Eval., Yorktown Heights, N.Y., 113-137 (Aug. 1977).
19. Graham,G.S., Denning,P.J. : On the relative controllability of memory policies. In Int. Symp. on Comp. Perf. Model., Meas., and Eval., Yorktown Heights, N.Y., 411-428 (Aug. 1977).
20. Parent,M., Potier,D. : A note on the influence of program loading on the page fault rate. In Model. and Perf. Eval. of Comp. Syst., Ispra, Italy, (Oct. 1976).
21. Brandwajn,A., Mouneix,B. : A study of a page-on-demand system. Information Processing Letters, 6, 125-132 (1977).
22. Scherr,A.L. : An analysis of time-shared computer systems. Cambridge (Mass.) : MIT Press 1967.
23. Abell,V.A., Rosen,S. : Performance of an ECS-based time-sharing subsystem. In Int. Symp. on Comp. Perf. Mod., Meas., and Eval., Yorktown Heights, N.Y., 249-261 (Aug. 1977).
24. Little,J.D. : A proof of the queueing formula L= λW. Operations Research 9, 383-387 (1961).
25. Baskett,F., Chandy,K.M., Muntz,R.R., Palacios,F.G. : Open,closed and mixed networks of queues with different classes of customers. J. ACM 22, 248-260 (1975).
26. Leroudier,J., Parent,M., : Quelques aspects de la modélisation des systèmes informatiques par simulation à événements discrets. R.A.I.R.O. Informatique 10, 5-26 (1976).
27. Fishman,G.S. : Statistical analysis for queueing simulations. Management Science 20, 363-369 (1973).
28. Chandy,K.M., Herzog,U., Woo,L. : Approximate analysis of queueing network models. IBM Research Report, RC 4931, Yorktown Heights, N.Y., July 1974.
29. Denning P.J., Buzen,J. : Operational analysis of queueing networks . In 3rd Int. Symp. on Model. and Perf. Eval. of Comp. Syst., Bonn, Germany, Oct. 1977.
30. Schwetman,H.D. : Hybrid simulation models : a speed-up technique combining analytic and discrete-event modeling. In Modelle fuer Rechensysteme, Workshop der GI, Bonn, Germany, Springer Verlag 9, 226-237 (April 1977).
31. Brandwajn,A. : Simulation de la charge d'un systeme conversationnel. R.A.I.R.O. Informatique 10, 25-40 (1976).
32. Brandwajn,A., Hernandez,J.A. : A study of a mechanism for controlling multiprogrammed memory in an interactive system. Research Report ENST-D-78008 Ecole Nationale Supérieure des Télécommunications, Paris, May 1978.

COMMUNICATION NETWORK MODELLING I

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

MODELLING OF LOCAL COMPUTER NETWORKS

O. Spaniol
Institut für Informatik
Universität Bonn
D - 5300 BONN (Germany)

This paper presents a new concept for local computer networks. Main features of the new model are: automatic reservations for colliding packets (thus avoiding repeated collisions); near optimum throughput; finite maximum waiting times even for priority scheduling policies.

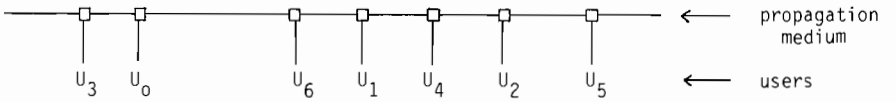
The performance of the system is analyzed by means of a Semi-Markov-process. Numerical results have been obtained for the most interesting parameters. A comparison with reservation techniques which have been proposed for other computer networks is given in the final section of the paper.

I. INTRODUCTION

Various multiplexing schemes (ALOHA, CSMA, Reservation schemes, Polling, ...) for packet switching networks have been investigated in the recent past (see [Sc]). Some of these methods have been designed for remote networks, but others are particularly effective for local computer networks which may be characterized by the fact that the propagation delay of the signal is small compared to the packet transmission time.

In 1976, a prototype of a new concept for local computer networks has been presented by Metcalfe and Boggs [MB]; this technique (called ETHERNET) is particularly attractive by reason of its structural simplicity. ETHERNET combines any number of stations by means of a logically passive medium for the propagation of packets. By reason of the limited distance between the users (less than 1 km) in the ETHERNET prototype a coaxial cable has been choosen as the propagation medium but other media could be used as well; for example, the extremely rapid development of fiber optics (see [Li]) suggests to choose fiber optics instead of coaxial cables. A system of this type (called FIBERNET) is actually in investigation (see [RM]). Since the transmission principles of FIBERNET are basically very similar to those used in ETHERNET we will restrict ourselves on ETHERNET in the following.

The structure of ETHERNET may be modelled as an unrooted tree; a new user joins the system simply by tapping into the propagation medium at any convenient point.



Packets in ETHERNET are of variable length; they include a source and a destination field in the header. Packet transmission is similar to the 'Carrier Sense Multiple Access' (CSMA) technique, i.e. most of the possible interferences are avoided by listening to the carrier. Thus a collision will occur only if the channel is erroneously sensed idle during the beginning of another packet transmission. Due to the limited distance between the users, a packet interference will be detected within a small fraction of a packet transmission time; the conflicting packets will be truncated, thus wasting only a small amount of channel time.

The transmission of collided packets has to be rescheduled. It is well known that uncontrolled systems will be overloaded sooner or later by an increasing number of interferences and retransmissions; for this reason, ETHERNET uses a heuristic

retransmission control policy which is based only on local information. The strategy attempts to relieve a temporarily overloaded system of a part of its load by introducing higher delays for packets which have been colliding more than once. The mean retransmission delay is determined by the degree of overloading which is estimated by the number of interferences of the packet in question.

Formally: a packet belongs to class i if it has already had i interferences. Then the probability f_i that a class i packet will retransmit is choosen as follows:

$$0 < f_i < 1 \qquad f_{i+1} < f_i \qquad \lim_{i \to \infty} f_i = 0 \qquad \text{(see [Ba],[BG])}.$$

It has been shown by simulations that ETHERNET may be controlled both by a polynomial increase of mean retransmission delay ($f_i = p/c \uparrow i$) or by an exponential increase of delay ($f_i = q/i \uparrow E$) if the parameters p,c,q and E are suitably choosen.

In addition to the necessity of controlling the retransmissions (uncontrolled systems are unstable!) the ETHERNET concept has several other structural shortcomings:

a. There is no upper bound for the maximum number of interferences of a packet.
b. Collided packets are discriminated by increasing delays (unfairness!).
c. High variations of waiting times due to this unfairness effect.

In this paper a modification of ETHERNET will be presented which tries to overcome these disadvantages by introducing a slotted local computer network (SLOTTED ETHERNET). This approach will maintain the principal advantages of ETHERNET but it will be impossible that a particular packet will collide more than once. This will be effected by reservations which are to be made in a 'Time Division Multiple Access' mode by the conflicting users immediately after seeing a collision. The retransmission of the packets will be automatically scheduled according to any priority scheme which has been agreed by the users. In this retransmission period it will be guaranteed by a special transmission protocol that no unauthorized packet transmission will take place; thus repeated collisions of a packet will be avoided.

This concept implies a finite maximum waiting time until success for any priority scheme; the maximum is proportional to the total number of users.

For exponentially distributed packet generating times the behaviour of SLOTTED ETHERNET may be interpreted as a Semi-Markov process. An evaluation of the throughput and the waiting times of the system will be given in sections III and IV. In the final section of the paper SLOTTED ETHERNET will be compared with other models.

The price to be paid for the advantages of the new concept is that variable packets are not allowed and that the system can only accept a finite total number of users which depends on the slot (i.e. packet) length and the maximum signal propagation delay. The first of these drawbacks may be overcome by introducing multipacket transmissions (see II.3) or by a BUSY TONE ETHERNET model (see Appendix). An increase of the number of users is possible either by extending the slot length or by introducing additional reservation slots, but these solutions would reduce the throughput and increase the overhead as well as the waiting times of the system.

II. THE SLOTTED ETHERNET CONCEPT

In this and the following sections we restrict on a slotted version of ETHERNET (i.e. fixed maximum length of a single packet); the problems arising with variable packet sizes will be discussed in the appendix.

II.1. DESCRIPTION OF THE MODEL

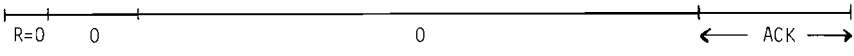
Due to the small amount of propagation delay in local networks, a collision will be detected very quickly; thus only a small amount of a slot will by wasted by a conflict. If a collision occurs in SLOTTED ETHERNET, then we assume that all users which are involved in this collision will notify its actual priority in a TDMA mode during the rest of this slot; each of these users listens to the channel during this "reservation slot" and schedules its retransmission slot according to a prearranged priority rule (see below). It will be assumed that the system consists of a fixed number, N, of users $U_0, \ldots, U_{N-1}$. A user is not necessarily active.

Each slot begins with a group R of bits announcing whether a retransmission is scheduled for this slot (R = 1) or not (R = 0). R = 1 will be given by the (previously collided) station whose retransmission will take place in this given slot.

A slot ends with an ACK part which confirms a successful transmission in the previous slot. Acknowledgments will not be considered in the model analysis because they are not susceptible to collisions.

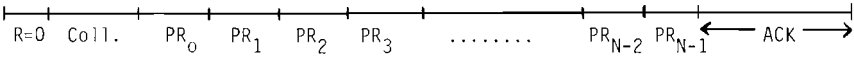
There are the following four types of slots which have to be distinguished:

a. "Free":



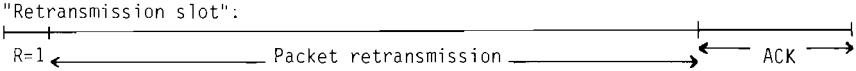
b. "Immediate success":

c. "Reservation slot":



(PR_i := priority of user U_i; for example: priority = actual waiting time)

d. "Retransmission slot":



Remarks:

a. The combination of ETHERNET and TDMA principles in SLOTTED ETHERNET removes all of the collisions before admitting new packet transmissions. Thus, no packet can collide more than once; this leads to a finite maximum waiting time until success since the total number of users is limited.

b. The maximum number, N, of users which can be tolerated by the system depends on the packet size (slot length); this number will be further reduced by the safe guards which are necessary between any two consecutive slot sections. If the number of users grows bigger and bigger, we could either increase the slot length or introduce additional reservation slots; both solutions will reduce the throughput of the system.

c. As a priority rule for user U_i we could choose for example:

$PR_i := a_i \cdot z + b_i$ (where $a_i, b_i \geq 0$ and z := actual waiting time in full slots).

As a tie breaking rule for equal priority values, the strategy "higher number first" or any other could be used.

As an example, the behaviour of a system with $PR_i := z$ and "higher numbered user first" is given in figure 1.

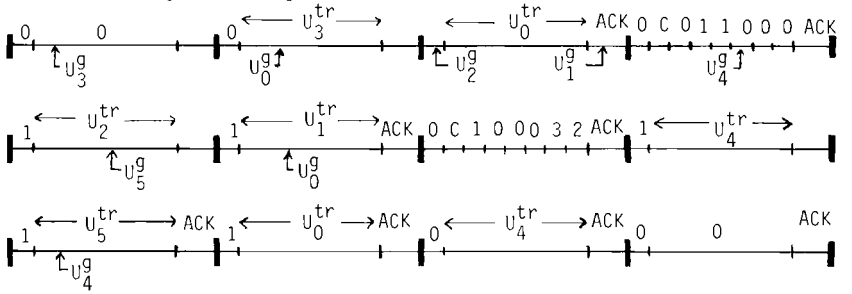


Figure 1: Transmissions in SLOTTED ETHERNET with 6 users $U_0, \ldots, U_5$.
($U_i^{tr} \triangleq U_i$ transmits successfully ; $U_i^{g} \triangleq U_i$ generates a new packet)

II.2. THE INTERPRETATION OF SLOTTED ETHERNET AS A SEMI-MARKOV PROCESS

The behaviour of SLOTTED ETHERNET may be described as a sequence of RETRANSMISSION PERIODS (RTP) $X_0, \ldots, X_i, X_{i+1}, \ldots$

where $X_i := RTP_j \stackrel{def}{=} j \iff$ the i-th retransmission period consists of j users

$\iff$ j packets have been generated during X_{i-1}.

The duration (in slots) of a retransmission period is given by

$$d(X_i) = \left\{ \begin{array}{ll} 1 & \text{if } X_i \leq 1 \\ X_i + 1 & \text{if } X_i \geq 2 \end{array} \right\} = X_i + \text{sign}(|X_i - 1|)$$

since no reservation will be necessary if only one packet will be generated in a retransmission period ("immediate success" in the next nonreserved slot). A free slot may be considered as a retransmission period which contains no users. Finally, if two or more users generate a new packet in a period, a reservation slot has to be added to the next retransmission period.

If the packet generating time of user U_i (i=0,..,N-1) is exponentially distributed with parameter λ_i then the stochastic process $\{X_n, n=0,1,2,\ldots\}$ is an aperiodic and irreducible Markow chain with finite state space $\{0,1,\ldots,N\}$.

Thus $\pi_i := \lim Pr(X_n=i)$ is given by the solution of the system of equations:

$[\pi \cdot P = \pi \quad , \quad \pi \cdot \vec{1} = 1]$

where $P := (p_{i,j})$ and $p_{i,j} := Pr(X_{n+1}=j | X_n=i)$ (see [Ci] and [Ro]).

If $[T_i, T_{i+1})$ is the time interval which belongs to X_i (i.e. $T_i = \sum_{r=0}^{i-1} d(X_r)$) then the continuous time stochastic process $\{Y_t, t \geq 0\}$

defined by $Y_t := X_n$ if $t \in [T_n, T_{n+1})$

is a SEMI-MARKOW PROCESS with sojourn time $d(r)$ in state r $(r = 0,\ldots,N)$. Sojourn times depend on the actual state, but are independent of the next state which will be visited.

It can be shown (see [Ci]) that

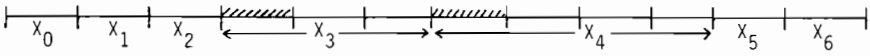
$P_i :=$ proportion of time that the stochastic process Y_t spends in state i

$$= \lim_{t\to\infty} Pr(Y_t=i) = \frac{\pi_i \cdot d(i)}{\sum_{r=0}^{N} \pi_r \cdot d(r)} = \frac{\pi_i \cdot d(i)}{\overline{\pi} + 1 - \pi_1}$$

where $\overline{\pi} := \lim_{n\to\infty} E(X_n) = \sum_{r=0}^{N} r \cdot \pi_r$ = expected number of users in a RTP.

Remark: In general, the formula given for P_i holds only for Semi-Markov-processes which are not lattice, but due to the simple structure of the process in SLOTTED ETHERNET (sojourn times in state r are fixed and independent of the next state to be visited) it can be shown that the formula can also be applied for our model.

The retransmission periods of figure 1 are demonstrated in figure 2.



i	0	1	2	3	4	5	6
X_i	0	1	1	2	3	1	0
$d(X_i)$	1	1	1	3	4	1	1

≙ reservation slot

Figure 2: Retransmission periods in SLOTTED ETHERNET

II.3. MULTIPACKET TRANSMISSION

SLOTTED ETHERNET can also be used for multipacket reservation and transmission. In this case, each packet in a retransmission period has to begin with $R = \alpha$ (instead of $R = 1$) where $\alpha \geq 1$ is the number of packets which have not yet been transmitted by the actual user; this holds also for the packets of an immediately successful transmission (except for the first packet). Thus, the actual user will finish his multipacket transmission after the following α slots; then the next user will be scheduled exactly as in SLOTTED ETHERNET for single packets.

With multipacket transmission facilities, the throughput of SLOTTED ETHERNET will be increased as well as the maximum waiting time for a particular user. The size of a multipacket should be limited in order that no user will eventually monopolize the system.

The throughput and waiting time considerations of sections III and IV are based on SLOTTED ETHERNET systems which are designed for single packet transmissions.

III. THROUGHPUT OF SLOTTED ETHERNET SYSTEMS

III.1. GENERAL THROUGHPUT FORMULA. LOWER AND UPPER BOUNDS

Let $\pi_i := \lim_{n\to\infty} \Pr(X_n=i)$ and $d(i) := \begin{cases} 1 & \text{if } i \leq 1 \\ i+1 & \text{if } i \geq 2 \end{cases}$

be the probability and the length of a retransmission period consisting of i users.

Then the throughput S of the system is given by:

$$S = \frac{\sum_{i=0}^{N} i\cdot\pi_i}{\sum_{i=0}^{N} d(i)\cdot\pi_i} = \frac{\bar{\pi}}{\bar{\pi} + 1 - \pi_1} = 1 - \frac{1-\pi_1}{\bar{\pi}+1-\pi_1}$$

where $\bar{\pi} := \sum_{i=0}^{N} i\cdot\pi_i$ denotes the mean length of a retransmission period.

The following lower and upper bounds for the throughput are easily obtained:

A. LOWER BOUND:

$$S \geq S_{min} = \frac{2\cdot(1-\pi_0)}{2(1-\pi_0)+1} = 1 - \frac{1}{2(1-\pi_0)+1} = \frac{2}{3} - \frac{2\pi_0}{9-6\pi_0}$$

with equality if and only if $\pi_2 = 1-\pi_0$ (i.e. zero or two arrivals per RTP).

B. UPPER BOUND:

$$S \leq S_{max} = 1 - \frac{1-\pi_1}{\pi_1 + N\cdot(1-\pi_1)} = \begin{cases} 1 & \text{if } \pi_1 = 1 \\ 1 - \dfrac{1}{\pi_1/(1-\pi_1) + N} & \text{if } \pi_1 < 1 \end{cases}$$

with equality if and only if $\pi_{N-1} = 1-\pi_1$ (i.e. one or N-1 arrivals per RTP).

The following extremes of S_{max} may be considered:

1. $S_{max}^{(1)} = 1$ if $\pi_1 = 1$ (i.e. exactly one arrival per slot, perfect scheduling)

2. $S_{max}^{(2)} = 1 - \frac{1}{N}$ if $\pi_{N-1}=1$ (overloaded system, all but the the last user generate a new packet during the RTP).

These bounds hold independently of the distributions of packet generating times.

Remark: If k reservation slots are used (due to an excessive number of users) then:

$$\frac{2}{2+k} - \frac{2\cdot\pi_0}{(2+k)\cdot((2+k)\cdot(1-\pi_0)+\pi_0)} \leq S = 1 - \frac{\pi_0+k\cdot(1-\pi_0-\pi_1)}{\bar{\pi}+\pi_0+k\cdot(1-\pi_0-\pi_1)} \leq 1 - \frac{k\cdot(1-\pi_1)}{(N+k-1)\cdot(1-\pi_1)+\pi_1} .$$

III.2. CALCULATION OF π_i FOR EXPONENTIALLY DISTRIBUTED PACKET GENERATING TIMES

We assume that the system is composed of N independent users and that the packet generating time for each user is exponentially distributed with mean $1/\lambda$, i.e.

$$\Pr(U_i \text{ generates a packet during } [t,t+h) \mid U_i \text{ active at time } t) = 1 - e^{-\lambda \cdot h}.$$

As already mentioned in section II, the sequence of retransmission periods is in this case an aperiodic and irreducible Markov chain with unique positive stationary state probabilities π_i (i=0,...,N).

In order to find the solution of the system of equations which determines these probabilities we have to calculate the transition probabilities $p_{i,j}$ given by:

$$p_{i,j} := p_{i,j}^{(N)} := \Pr(X_{n+1}=j \mid X_n=i) =$$
$$= \Pr(j \text{ of the N users generate a new packet during a RTP of i users}).$$

These values are obtained as follows:

A. i = 0 :

$$p_{0,j}^{(N)} = \Pr(\text{ j packets are generated in a free slot }) = \binom{N}{j} \cdot (1 - e^{-\lambda})^j \cdot (e^{-\lambda})^{N-j} \qquad (j = 0,\ldots,N).$$

B. i = 1 :

$$p_{1,j}^{(N)} = \Pr(\text{ j of the N-1 active users generate in } RTP_1) = \binom{N-1}{j} \cdot (1-e^{-\lambda})^j \cdot (e^{-\lambda})^{N-1-j} = p_{0,j}^{(N-1)} \qquad (j = 0,\ldots,N).$$

C. i ≥ 2 :

The retransmission period consists of one reservation slot followed by i transmission slots.
Since the transition probabilities are not influenced by the numbering of the users, we assume that the RTP consists of the users U_r (r=0,...,i-1) and that the transmissions will be scheduled according to the sequence $U_{i-1}, U_{i-2}, \ldots, U_0$.

It will be assumed that a blocked user will become active again (i.e. he starts a new packet generation time) immediately after the transmission of his packet. Thus the production period of user U_r in RTP_i consists of

$$\begin{cases} i+1 \text{ slots} & \text{if } r = i,i+1,\ldots,N-1 \\ r \text{ "} & \text{if } r = 0,1,\ldots,i-1 \end{cases}.$$

Let $G(r,i) := \Pr(U_r \text{ doesnot generate a new packet during } RTP_i)$.

Then: $$G(r,i) = \begin{cases} e^{-\lambda \cdot (i+1)} & \text{if } r \geq i \\ e^{-\lambda \cdot r} & \text{if } r < i \end{cases}.$$

If U_r^1 denotes the event that U_r produces a new packet, and if U_r^0 is the event that no new packet will be produced by U_r then:

$$\Pr(U_r^{\varepsilon_r}) = \begin{cases} 1 - G(r,i) & \text{if } \varepsilon_r = 1 \\ G(r,i) & \text{else} \end{cases} \Bigg\} = \varepsilon_r - (2 \cdot \varepsilon_r - 1) \cdot G(r,i).$$

The transition probabilities $p_{i,j}^{(N)}$ are now given by:

$$p_{i,j}^{(N)} = \sum_{\{(\varepsilon_0,\ldots,\varepsilon_{N-1}) \in B^N \mid \sum_{i=0}^{N-1} \varepsilon_i = j\}} \Pr(U_0^{\varepsilon_0} \wedge U_1^{\varepsilon_1} \wedge \ldots \wedge U_{N-1}^{\varepsilon_{N-1}})$$
$$= \sum_{\{(\varepsilon_1,\ldots,\varepsilon_{N-1}) \in B^{N-1} \cdot \sum_{i=1}^{N-1} \varepsilon_i = j\}} \Pr(U_1^{\varepsilon_1}) \cdot \Pr(U_2^{\varepsilon_2}) \cdot \ldots \cdot \Pr(U_{N-1}^{\varepsilon_{N-1}}).$$

The system of equations which determines the stationary state probabilities π_i is now easily solved; the solution depends only of the parameters λ and N.

Figure 3 shows the trade-off between throughput S and the distribution parameter λ for several values of N.

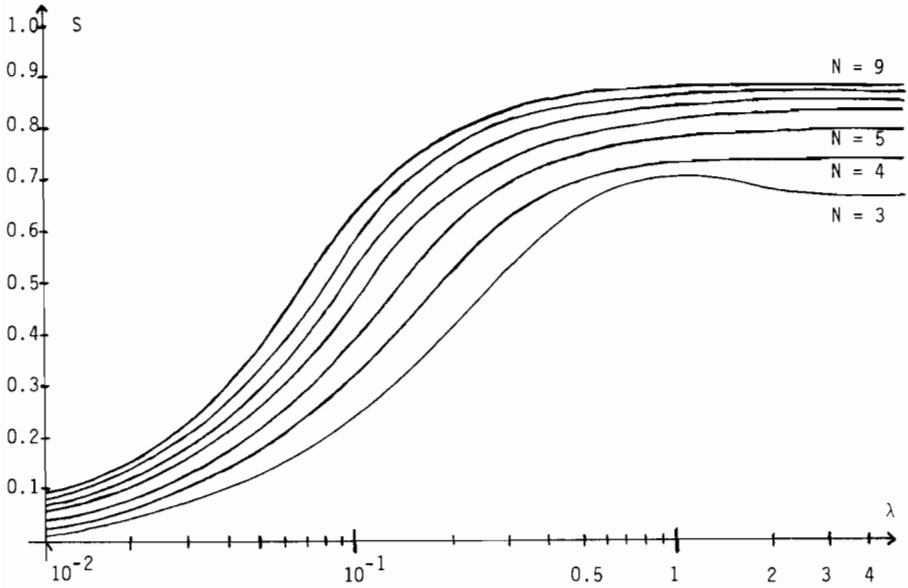


Figure 3: Troughput of SLOTTED ETHERNET systems (exponentially distributed packet generating times; parameter λ)

III.3. ASYMPTOTIC BEHAVIOUR OF THROUGHPUT

The throughput of SLOTTED ETHERNET systems may be asymptotically calculated for very small and for very large packet generating rates:

Theorem: a. $S \xrightarrow[\lambda\to\infty]{} S_{max}^{(2)} = 1 - \frac{1}{N}$.

b. If packet gener. times are expon. distr. with parameter λ then:

$$S \xrightarrow[\lambda\to 0]{} \pi_1 \to \frac{N\cdot\lambda}{1+\lambda} \leq N\cdot\lambda .$$

Proof:

a. If $\lambda\to\infty$ then $\pi_{N-1} \to 1$. Thus $S \to S_{max}^{(2)}$ (see III.1).

b. If $\lambda\to 0$ then $p_{i,j} \to 0$ for $i>1$ or $j>1$; thus from III.2 we obtain:

$$p_{0,0}^{(N)} \to e^{-\lambda\cdot N}, \quad p_{0,1}^{(N)} \to 1 - e^{-\lambda\cdot N}, \quad p_{1,0}^{(N)} \to e^{-\lambda(N-1)}, \quad p_{1,1}^{(N)} \to 1 - e^{-\lambda(N-1)}.$$

Thus, disregarding terms of order λ^2 or higher, the system of equations for the stationary probabilities reduces to:

$$\pi_0 = \pi_0\cdot p_{0,0} + \pi_1\cdot p_{1,0} \quad , \quad \pi_0 + \pi_1 = 1 .$$

Thus $S = \pi_1 = \frac{1 - p_{0,0}}{1-p_{0,0}+p_{1,0}} \xrightarrow[\lambda\to 0]{} \frac{1 - (1-\lambda N)}{1 - (1-\lambda N) + (1-\lambda(N-1))} = \frac{\lambda\cdot N}{1+\lambda}$.

IV. WAITING TIME ANALYSIS

IV.1. UPPER BOUNDS

A main advantage of SLOTTED ETHERNET is the maximum waiting time until success; furthermore, the waiting time is upper bounded for any priority scheme:

Theorem: a. FIFO-scheduling: $W_{max} = N+1$ slots

b. Arbitrary priority scheme: $W_{max} = 2 \cdot N-1$ slots.

Proof:

a. In a FIFO sequencing rule, a customer will be delayed by at most the transmission of all other users and by a reservation slot; an additional slot of waiting time will be encountered if the packet has been generated immediately after the beginning of a slot. Thus: $W_{max} = N-1 +1 +1 = N+1$ slots.

b. In the worst case, a user will generate his packet at the beginning of the reservation slot of a RTP which consists of all the other users. If the next RTP also contains N-1 users and if (due to a very discriminating priority rule) the transmission of our particular user will be scheduled after all these other packet transmissions, then: $W_{max} = 1 + N-1 + 1 + N-2 = 2 \cdot N-1$ slots.

IV.2. MEAN WAITING TIMES

In this section, mean waiting times for any priority scheme will be calculated. It will be assumed that packet generating times are exponentially distributed with parameter λ.

The actual waiting time $W^{(i,j)}$ of a user whose packet is generated in RTP_i ($i \geq 0$) and whose packet transmission will take place in RTP_j ($j > 1$) is composed of three parts:

$$W^{(i,j)} = W_1^{(i)} + W_2^{(j)} + W_3^{(j)}$$

where $W_1^{(i)}$ is the waiting time in the generation period RTP_i: $0 \leq W_1^{(i)} \leq i+1$.

$W_2^{(j)}$ is the waiting time which is caused by an eventual reservation slot in RTP_j (which will be necessary if and only if at least one of the other j users did produce a packet in the same period RTP_i).

Thus $W_2^{(j)} = \begin{cases} 1 & \text{if } j \geq 2 \\ 0 & \text{if } j = 1 \end{cases} = \text{sign}(j-1)$.

$W_3^{(j)}$ is the waiting time in RTP_j (until the beginning of the packet transmission). $0 \leq W_3^{(j)} \leq j-1$.

Mean waiting times are obtained as follows:

In the mean, the (re-)transmission of a user will be scheduled after (j-1)/2 transmissions of his competitors.

Thus: $\overline{W_3^{(j)}} = \frac{j-1}{2}$ (this holds for any priority scheme).

$\overline{W_1^{(i)}}$ depends on the packet generating time distribution. For an exponential distribution, the mean can easily be calculated:

Lemma: If packet generating times are exponentially distributed with mean $1/\lambda$ then:

a. $\overline{W_1^{(0)}} = \overline{W_1^{(1)}} = 1 - \frac{1}{\lambda} + \frac{1}{e^{\lambda}-1} \in [\frac{1}{2} : 1]$.

b. $i \geq 2$:

$$\overline{W_1^{(i)}} = \left[(N-i) \cdot f(i+1) + \sum_{h=1}^{i-1} f(h)\right] / \left(N-1 - (N-i) \cdot e^{-\lambda \cdot (i+1)} - \sum_{h=1}^{i-1} e^{-\lambda \cdot h} \right)$$

where $f(h) := h - \frac{1}{\lambda} \cdot (1 - e^{-\lambda \cdot h})$.

Proof:
Assume that U_r generates a packet in RTP_i which is assumed to be composed of the users $U_{i-1},\ldots,U_0$ (in this order).

Let $\overline{W_{1,r}^{(i)}}$:= expected waiting time of U_r in RTP_i.

$E(r,i)$:= expected packet production instant relative to the beginning of RTP_i.

$$M(r,i) := \begin{cases} 1 & \text{if } i \le 1 \\ r & \text{if } i \ge 2 \text{ and } r < i \\ i+1 & \text{if } i \ge 2 \text{ and } r \ge i \end{cases}$$

Thus $\overline{W_{1,r}^{(i)}} = M(r,i) - E(r,i)$ for $i = 0,\ldots,N$ (see III.2).

$E(r,i)$ is calculated as follows:

a. $i \le 1$:
RTP_i consists of exactly one slot. If Z_r is the production instant of U_r, then:

$$Pr(Z_r \le h \mid U_r \text{ produces in } RTP_0 \text{ or in } RTP_1) = Pr(Z_r \le h \mid Z_r \le 1) = \frac{Pr(Z_r \le h)}{Pr(Z_r \le 1)} = \frac{1 - e^{-\lambda \cdot h}}{1 - e^{-\lambda}} \quad (h \le 1).$$

Thus: $E(r,0) = E(r,1) = \int_0^1 \frac{x \cdot \lambda \cdot e^{-\lambda \cdot x}}{1 - e^{-\lambda}}\, dx = \frac{1}{\lambda} - \frac{1}{e^{\lambda}-1} \in [0 : \frac{1}{2}]$.

b. $i \ge 2$:
The activity period of U_r in RTP_i has a duration of $M(r,i)$ slots. Thus:

$$Pr(Z_r \le h \mid U_r \text{ produces in } RTP_i) = Pr(Z_r \le h \mid Z_r \le M(r,i)) = \frac{1 - e^{-\lambda \cdot h}}{1 - e^{-\lambda \cdot M(r,i)}}$$

Hence: $E(r,i) = \int_0^{M(r,i)} \frac{x \cdot \lambda \cdot e^{-\lambda \cdot x}}{1 - e^{-\lambda \cdot M(r,i)}}\, dx = \frac{1}{\lambda} - \frac{M(r,i)}{e^{\lambda \cdot M(r,i)}-1}$.

If $Q(r,i)$ is the probability that a customer producing in RTP_i is U_r, then:

$$Q(r,i) = (1 - G(r,i)) / \sum_{h=0}^{N-1} (1-G(h,i)) = (1 - G(r,i)) / \sum_{h=1}^{N-1} (1-G(h,i))$$

where $G(r,i)$ is the probability that U_r produces a new packet in RTP_i (see III.2).

Thus: $\overline{W_1^{(i)}} = \sum_{r=0}^{N-1} Q(r,i) \cdot \overline{W_{1,r}^{(i)}} = \sum_{r=1}^{N-1} Q(r,i) \cdot (M(r,i) - E(r,i))$

which after insertion of the formulae reduces to the expression given in the lemma.

The mean waiting time $\overline{W}$ of an arbitrary packet is now obtained as follows:

Theorem: $\overline{W} = \frac{1}{2} + \sum_{i=0}^{N} P_i \cdot [\overline{W_1^{(i)}} + \frac{1}{2} \cdot (\sum_{j=2}^{N} j \cdot p_{i,j}^{(N)} - p_{i,1}^{(N)}) / (1-p_{i,0}^{(N)})]$

where the values $\overline{W_1^{(i)}}$ are obtained from the previous lemma (for exponentially distributed packet generating times).

Proof:
SLOTTED ETHERNET may be modelled as a Semi-Markow-process (see II.2).
Thus an arbitrary packet will be produced in RTP_i with probability

$$P_i = \pi_i \cdot d(i) / (\overline{\pi} + 1 - \pi_1).$$

The packet transmission will be scheduled in RTP_j $(j \ge 1)$ with probability $P_i \cdot p_{i,j}^{(N)} / (1-p_{i,0}^{(N)})$. Thus $\overline{W}$ is given by:

$$\overline{W} = \sum_{i=0}^{N} \sum_{j=1}^{N} (\overline{W_1^{(i)}} + \text{sign}(j-1) + \frac{j-1}{2}) \cdot P_i \cdot p_{i,j}^{(N)} / (1-p_{i,0}^{(N)}).$$

Q.E.D.

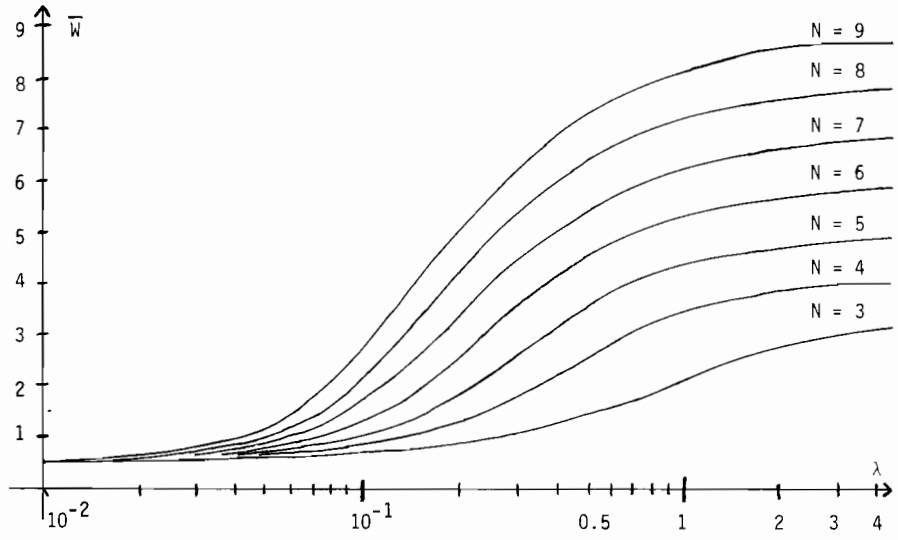


Figure 4: Mean waiting times of SLOTTED ETHERNET systems (exponentially distributed packet generating times; parameter λ)

IV.3. BOUNDS AND ASYMPTOTIC BEHAVIOUR OF MEAN WAITING TIMES IN SLOTTED ETHERNET

The following relations for $\overline{W}$ are obtained for a given number, N, of users of the SLOTTED ETHERNET system:

Theorem:

a. $\overline{W}$ is monotonically increasing with λ. The mean waiting time is bounded by:

$$\frac{1}{2} = \overline{W}(\lambda=0) \leq \overline{W} \leq N + \frac{1}{N-1} = \overline{W}(\lambda=\infty) .$$

b. The following approximations of $\overline{W}$ are obtained:

$$\overline{W} \sim \frac{1}{2} + \frac{3}{4}\cdot(N-1)\cdot\lambda \quad \text{if} \quad (N-1)\cdot\lambda << 1 \quad \text{(exponential distribution)}$$

$$\overline{W} \sim N + \frac{1}{N-1} - \frac{1}{\lambda} \quad \text{if} \quad (N-1)\cdot\lambda >> 1 \quad \text{(any distribution of packet generating times)}$$

Both approximations become exact if $\lambda \to 0$ and $\lambda \to \infty$ respectively.

Proof:

a. The mean waiting time is monotonically increasing with λ since an increase of the packet generation rate will result in more and in earlier packet productions. Thus the waiting time in the production period RTP_i will increase as well as the waiting time in the retransmission period RTP_j ; furthermore, the probability of a collision will be increasing.

To show that $\overline{W} \in [\frac{1}{2} : N + \frac{1}{N-1}]$ it will be sufficient to verify the approximations given in part b. of the theorem.

b. Small λ; exponential distribution:

It is easily verified that:

$P_0 = 1 - N\cdot\lambda + O(\lambda^2)$, $P_1 = N\cdot\lambda + O(\lambda^2)$, $P_i = O(\lambda^2)$ for $i > 1$.

Thus: $\overline{W} = \frac{1}{2} + \sum_{i=0}^{1} P_i \cdot [\overline{W_1^{(i)}} + \frac{1}{2} \cdot (2 \cdot p_{i,2}^{(N)} - p_{i,1}^{(N)})/(1-p_{i,0}^{(N)})] + O(\lambda^2)$

$= 1 + \frac{1}{2} \cdot \sum_{i=0}^{1} P_i \cdot \frac{2 \cdot p_{i,2} - p_{i,1}}{1-p_{i,0}} + O(\lambda^2)$.

Now: $\frac{2 \cdot p_{0,2} - p_{0,1}}{1-p_{0,0}} = \frac{2 \cdot \binom{N}{2} \cdot \lambda^2 - N\lambda(1-(N-1)\lambda)}{N\lambda - \binom{N}{2} \cdot \lambda^2} + O(\lambda^2)$

$= (2 \cdot (N-1) \cdot \lambda - 1)(1 + \frac{N-1}{2}\lambda) + O(\lambda^2) = \frac{3}{2} \cdot (N-1) \cdot \lambda - 1 + O(\lambda^2)$

and: $\frac{2 \cdot p_{1,2} - p_{1,1}}{1-p_{1,0}} = \frac{3}{2} \cdot (N-2) \cdot \lambda - 1 + O(\lambda^2)$.

Thus: $\overline{W} = 1 + \frac{1}{2} \cdot [(\frac{3}{2} \cdot (N-1)\lambda - 1)(1-N\lambda) - N\lambda] + O(\lambda^2) = \frac{1}{2} + \frac{3}{4}(N-1)\lambda + O(\lambda^2)$.

Large λ; arbitrary distribution:

If $\lambda \to \infty$ then: $P_{N-1} \to 1$, $p_{N-1,N-1}^{(N)} \to 1$, $\overline{W_2^{(N-1)}} \to 1$, $\overline{W_3^{(N-1)}} \to \frac{N-2}{2}$

(all retransmission periods consist of N-1 users; each RTP will contain a reservation slot; the actual user will be scheduled after (N-2)/2 of his competitors).

Thus $\overline{W} \to P_{N-1} \cdot (\overline{W_1^{(N-1)}} + 1 + \frac{N-2}{2}) \to \overline{W_1^{(N-1)}} + \frac{N}{2}$.

Furthermore:

$$Q(r,N-1) \to \begin{cases} 1/(N-1) & \text{if } r > 0 \\ 0 & \text{if } r = 0 \end{cases} \quad , \quad E(r,N-1) \to \frac{1}{\lambda} \ .$$

Hence: $\overline{W_1^{(N-1)}} = \sum_{r=1}^{N-1} Q(r,N-1) \cdot (M(r,N-1) - E(r,N-1)) \to \frac{1}{N-1} \cdot (1 + \sum_{r=1}^{N-1} r) - \frac{1}{\lambda}$

$= \frac{N}{2} + \frac{1}{N-1} - \frac{1}{\lambda}$.

The calculation of $\overline{W_1^{(N-1)}}$ for overloaded systems ($(N-1)\lambda \gg 1$) is illustrated in figure 5.

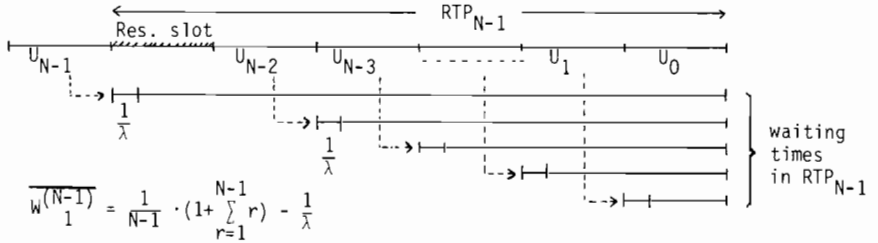


Figure 5: Mean waiting times in the production period RTP_{N-1} (overloaded systems)

V. COMPARISON OF SLOTTED ETHERNET WITH OTHER SLOTTED NETWORKS

V.1. THE ALOHA RESERVATION SYSTEM PROPOSED BY ROBERTS

As in the ALOHA reservation scheme (cf. [Rb]) SLOTTED ETHERNET alternates between a RESERVATION state and a RESERVED state. The throughput of both models is considerably higher than the throughput of systems without reservation (cf. [Ab], [Ba], [To]). It should be pointed out that Roberts reservation scheme suffers from the same instability problems as slotted ALOHA systems without reservations; thus a retransmission control policy will be necessary.

The main differences of Roberts system and SLOTTED ETHERNET are:

RESERVATION SCHEME	*SLOTTED ETHERNET*
Designed for remote networks: Propagation delay is much higher than packet transmission time	*Designed for local networks: Propagation delay is very small compared to the packet transmission time*
System turns to the RESERVED state after seeing a successful reservation	*The RESERVED state is preceded by a collision (reservation slot)*
The number of RESERVED slots between two RESERVATION slots is a constant	*Number of RESERVED slots is given by the number of colliding users*
The throughput will be reduced by the RESERVATION slots	*The throughput is not necessarily reduced by reservation slots (see III.1)*
No upper bound for the maximum number of collisions of a given packet	*A packet can collide at most once.*
Any number of users is allowed to be in the system simultaneously	*The maximum number of admissible users depends on the slot length; it can be increased by introducing additional reservation slots*

V.2. COMPARISON WITH TDMA AND FDMA SCHEMES

SLOTTED ETHERNET systems are more flexible than 'Time Division' or 'Frequency Division' schemes since the channel will never be idle if there are packet transmissions to be scheduled (except for the waiting time between the generation time and the beginning of the next slot). In contrast to TDMA and FDMA systems, SLOTTED ETHERNET is particularly suitable for bursty users.

V.3. THE IMPLICIT RESERVATION SCHEME

This reservation method for ALOHA type systems has been introduced by Banh, Gelenbe and Labetoulle ([Ba], [BG], [BL]). Its main characteristics are as follows:

After generating a multipacket, a user will first enter a waiting state where the system will be observed for L successive slots (L = window size). Some of these slots will be eventually free of a successful transmission (i.e. nonreserved) and the user will try to transmit the first packet of his multipacket in one of these supposedly free slots in the next cycle (modulo L). If the packet enters in a collision, then the next attempt to transmit will take place after a random retransmission delay in a nonreserved slot. On the contrary, if the packet has been successful, then beginning with this slot every L-th slot will be implicitly reserved for the remaining packets of this user. Thus the first successful transmission will seize one particular slot of the window for this user.

Using similar assumptions and arguments as with slotted ALOHA systems it can be shown that the throughput of the Implicit Reservation scheme is approximately given by:

$$S \approx \frac{\alpha}{e + \alpha - 1}$$

(α := expectation of the multipacket size, i.e. number of single packets per multipacket).

The throughput of the Implicit Reservation Scheme approaches unity if the mean size of a multipacket tends to infinity. But in this case the system will be monopolized by at most L users whose transmissions will never be stopped. This effect leads necessarily to a large variation of waiting times in heavily loaded systems since perhaps some of the users will be immediately successful whereas the others have to wait many slots until the first successful transmission.

It should be noted that the Implicit Reservation scheme has to be stabilized by a retransmission control strategy.

Implicit reservation schemes and SLOTTED ETHERNET are contrary to each other in many respects. Some of the main differences are illustrated in the following table:

IMPLICIT RESERVATION	*SLOTTED ETHERNET*
Substantial amount of propagation delay (but propagation delay ≤ window size L)	*Designed for local computer networks: very small propagation delay*
Implicit reservations after the first successful transmission	*(Implicit) reservation for single packets after a collision; can be extended on multipackets; see II.3.*
Before the first transmission, the channel is observed for L consecutive slots	*The (first) packet will be transmitted in the first slot which follows the actual retransmission period*
'Time Division' after the first success	*Priority scheduling after collisions*
Any number of users may be active simultaneously	*The maximum number of users depends on the slot length*
No upper bound for the maximum number of collisions, but no more collisions after the first successful transmission	*At most one collision per (multi-) packet; the maximum waiting time is upper bounded for any priority scheme*
Large variation of waiting time in heavily loaded systems	*Very small variation of waiting times even for heavily loaded systems*

VI. CONCLUSION

The rapid development of new transmission technologies (see [Li]) during the past few years has considerably increased the interest in the study of local computer networks. In this paper a new model for such networks has been presented which is based on the ETHERNET and on the FIBERNET concept. In contrast to these networks, the performance of SLOTTED ETHERNET has been evaluated by mathematical methods. Some other important features (finite maximum waiting times, priority scheduling policies, ...) indicate that SLOTTED ETHERNET is really an extension of ETHERNET.

The model can be extended to include multipacket transmissions or packets of variable length (see Appendix). These and various other model extensions are actually in investigation.

APPENDIX

If we want to generalize SLOTTED ETHERNET to include packets of variable length we are immediately confronted with the following problem:

Idle times in a reservation slot (i.e. time slices reserved for users which do not participate in a collision) and safety guards separating two packet (re-)transmissions could be sensed idle and erroneously be interpreted as unused.

Two proposals for a solution of this problem can be discussed:

I. The channel is sensed for a considerably longer time period (at least for a whole 'reservation slot'). A new packet is started if the channel is sensed idle during the whole carrier sense period.

   This solution leaves the equipment cost unchanged but results in some reduction of channel occupancy as well as in an increase of waiting times (especially for lowly loaded systems).

II. BUSY TONE ETHERNET:
   Transmitting users send a 'Busy Tone' (BT) over an additional channel during their transmission or reservation as well as during the following signal propagation delay. A new packet is started if the BT-channel is sensed idle.

   This method is an application of the Busy-Tone solution for 'Carrier Sense Multiple Access' (CSMA) techniques proposed by Tobagi (see [To]).

In both cases, retransmissions are to be scheduled exactly as in SLOTTED ETHERNET systems; thus the advantages of SLOTTED ETHERNET over ETHERNET are preserved.

The scheduled retransmission starts when the transmission channel is sensed idle.

Remarks:

a. No reservation indication is needed (this information is replaced by the longer period of carrier sensing and by the 'Busy-Tone'-channel respectively).

b. A main disadvantage of variable packets (compared to packet transmissions in SLOTTED ETHERNET) is that collided users have to be in a 'Busy Waiting' state of unknown duration until their retransmission begins. A remedy for this problem would consist in the indication of the priority as well as the duration of the planned packet transmission; these informations would have to be given by each user who participates in a collision.

REFERENCES

[Ab] N. Abramson: Packet Switching with Satellites. AFIPS Conference Proceedings 42 (1973) 695-702.

[Ba] T.A. Banh: Réseaux d'Ordinateurs à Commutation de Paquets: Modélisation Mathématique et Simulation en vue de l'Optimisation des Performances. Thèse de Doctorat en Sciences Appliquées. Université de Liège 1978.

[BG] T.A. Banh, E. Gelenbe: Performance Evaluation of Adaptive Policies for the Broadcast Channel. Proceedings of the Workshop on Data Communications, IIASA, Vienna (1975) 17-31.

[BL] T.A. Banh, J. Labetoulle: A Model of the ALOHA System with Reserved Slots. Rapport de Recherche 76-3, Service d'Informatique, Université de Liège (1976).

[Ci] E. Cinlar: Introduction to Stochastic Processes. Prentice Hall, Englewood Cliffs, New Jersey (1975).

[Li] T. Li: The Future of Optical Fibers for Data Communications. Proceedings of Fifth Data Communication Symposium, Snowbird, Utah (1977) 5-1 - 5-6.

[MB] R.M. Metcalfe, D.R. Boggs: ETHERNET: Distributed Packet Switching for Local Computer Networks. Communications of the ACM 19 (1976) 395-404.

[RM] E.G. Rawson, R.M. Metcalfe: FIBERNET: Multimode Optical Fibers for Local Computer Networks. IEEE Transactions on Communications COM-26 (1978) 983-990.

[Rb] L.G. Roberts: Dynamic Allocation of Satellite Capacity through Packet Reservation. AFIPS Conference Proceedings 42 (1973) 711-716.

[Ro] S.M. Ross: Applied Probability Models with Optimization Application. Holden-Day, San Francisco (1970).

[Sc] M.O. Scholl: Multiplexing techniques for Data Transmission over Packed Switched Radio Systems. Ph.D.Thesis, University of California, Los Angeles, UCLA-ENG-76123, (1976).

[To] F.A. Tobagi: Random Access Techniques for Data Transmission over Packet Switched Radio Networks. Ph.D.Thesis, University of California, Los Angeles, UCLA-ENG-7499, (1974).

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

A COMMUNICATION PROTOCOL AND A PROBLEM OF COUPLED QUEUES

L. BOGUSLAVSKII
Institute of Control Problems
USSR Academy of Sciences
Profsoyuznaya ul. 81
Moscow, USSR

E. GELENBE
Laboratoire de Recherche en Informatique
ERA 452 du Centre National de la Recherche Scientifique
Université Paris-Sud (Orsay)
91405 Orsay, France

We consider a two-way communication protocol used on a full-duplex link to transmit variable packets (or messages) between two communicating nodes or centers. Node 1 can transmit up to W_1 packets before obtaining an acknowledgement packet from Node 2 ; the corresponding quantity for Node 2 is W_2. When both nodes have attained this maximum transmission capacity (or window), they exchange acknowledgements before the transmission routine begins once again. If a node has no packet to transmit it sends a packet signalling its idle state to the other node in order to avoid unnecessary waiting times for the acknowledgement. This protocol is a generalization of the usual Send-and-Wait procedure and constitutes an alternative to protocols derived from the HDLC procedure. After introducing the protocol, we evaluate its throughput using a mathematical model for which an exact solution is provided. The problem we examine is equivalent to that of two finite capacity queues whose service is trigerred simultaneously when both queues attain their maximum capacity.

Nous étudions une procédure de communication entre deux noeuds permettant l'échange simultané de paquets ou de messages de longueur variable. Le Noeud 1 transmet W_1 paquets avant de s'arrêter pour attendre un acquittement du Noeud 2, qui peut envoyer W_2 paquets avant d'avoir besoin de recevoir un acquittement provenant de son partenaire. Il est convenu qu'un noeud n'ayant pas de paquets à transmettre le signale à l'autre noeud par un paquet spécial qui permet d'éviter une attente inutile avant l'échange d'acquittements. Cet échange a lieu quand les deux noeuds ont atteint le nombre maximum de paquets qu'ils peuvent envoyer. Cette procédure de liaison (ou protocole) est une alternative à des procédures obtenues à partir de HDLC. Nous présentons une évaluation de cette procédure en utilisant un modèle mathématique qui nous permet de calculer de façon explicite son débit maximum. Nous montrons que le problème étudié est équivalent à celui de deux files dont le service s'effectue simultanément quand les deux files ont atteint leur capacité de stockage maximum : la solution de notre problème permet donc de résoudre également un nouveau problème de files d'attente couplées.

INTRODUCTION

Protocols are procedures designed to guarantee that two or more processes exchanging messages or packets of data, can be informed of the proper receipt of the packets they send, or of transmission errors or failures. They are introduced at all significant levels of communicating systems and their structure has been defined in certain cases so as to obey international standards. A well-known example is the High- Level - Data - Link - Control (HDLC) procedure [6].

These procedures have an important effect on the performance of data communication systems : it has been shown [1,4] that the choice of protocol parameters conditions the throughput achievable by a data transmission link.

The purpose of this paper is to introduce a direct generalization of the Send - and - Wait (SW) protocol, which we shall call SW (W_1,W_2). SW governs the exchange of packets between two communicating nodes (N1 and N2) as follows. Consider a full-duplex link between N1 and N2 so that each node can send a packet to the other node independently of transmissions in the reverse direction. SW requires each node to stop its transmission, and to wait until the packet it has sent is acknowledged by a special packet (which we shall call the ACK), before it may transmit its next packet. The behaviour of the two communicating nodes controlled by SW can be represented by the state transition diagram shown on Figure 1. In state PP both nodes are transmitting a packet. States PW or WP indicate that one node is still transmitting a packet while the other has finished its transmission and is waiting for an ACK. In the state AA the two nodes exchanges ACKs.

In the SW(W_1,W_2) protocol, N1 can send up to W_1 packets to N2 before having to stop to wait for an ACK. The reception of the ACK will signify that all W_1 packets have been correctly received by N2, and the behaviour of N2 is symmetrical except that it can send up to W_2 packets before having to stop to wait for an ACK.

In Section 2 we will analyze this protocol. A mathematical model of its saturated performance will be presented and an explicit expression for the throughput in each direction will be obtained. We will then show that the problem considered is analogous to one of two coupled queues with finite capacity W_1 and W_2, and provide an analytical solution to this problem yielding the stationary joint queue length distribution. This model and result are new in the queueing theory context.

The performance of protocols has received considerable attention recently. In [1], simulation studies showing the incidence of protocols on the behaviour of computer network nodes are presented. A study of SW, using both analytical modelling and measurements, is presented in [4]. The choice of the optimum value of the time-out in host-to-host protocols is discussed in [2]. An analysis of HDLC [6] with and without packet losses is given in [5].

A MODEL OF THE SW(W_1,W_2) PROTOCOL

The behaviour of the SW(W_1,W_2) procedure will be represented, as was the case with SW, by a state transition diagram. We shall assume saturated behaviour : both N1 and N2 have an unlimited queue of packets to send to their partner. Thus our analysis will allow us to determine the maximum capacity of the SW(W_1,W_2) protocol, since it has been shown [1] that the throughput of a saturated queueing system is identical to the maximum arrival rate it can support before saturation.

The state of the protocol will be represented by an integer pair (i,j), $0 \leq i \leq W_1$, $0 \leq j \leq W_2$, where this state indicates that :

- if $i < W_1$ and $j < W_2$, N1(N2) has completed the transmission of i(j) packets and is currently transmitting the i+1 th(j+1 th),

- if $i=W_1$ and $j<W_2$ (or $i<W_1$ and $j=W_2$), N1 (or N2) has completed its transmission of W_1 (or W_2) packets and is waiting for an ACK,

- if $i=W_1$ and $j=W_2$, N1(N2) has received $W_1(W_2)$ packets is sending an ACK to N2 (N1). When both N1 and N2 have received an ACK, the state will return to (0,0) and the packet transmission procedure will begin once again.

The transitions between these states are shown on Figure 2. When both N1 and N2 have transmitted the maximum number of packets which they are allowed, they enter state (W_1,W_2) in which they exchanges ACKs.

Related queueing problem

The protocol model we have presented above is closely related to the following queueing problem.

Consider two queues Q1, Q2 of finite capacity W_1 and W_2 , respectively. Let i,j denote the length of Q1 and Q2, respectively. Assume that when $i=W_1$ all arrivals to Q1 are lost, the same being true for Q2 when $j=W_2$. The queueing system is such that no service is rendered as long as either $i<W_1$ or $j<W_2$. However, as soon as $i=W_1$ and $j=W_2$ the queues are emptied in a time S (which can be a function of W_1 and W_2). During the service time arrivals to both queues are lost : equivalently we may assume that both queues are instantaneously emptied at the end of the service epoch.

Notice that the state transition behaviour of the two queues will be identical to that of the protocol in all respects. The end of a packet transmission will correspond to an arrival to a queue. The exchange of ACKs in state (W_1,W_2) of the protocol corresponds to the service in state (W_1,W_2) in the coupled queueing system. The coupling of the queues only takes place when both queues are full ; the queues behave independently at other instants. These two queues may be visualized as two reservoirs of finite capacity which are emptied as soon as both are completely full.

ANALYSIS OF THE $SW(W_1,W_2)$ PROTOCOL AND OF THE COUPLED QUEUES

We shall state the assumptions concerning the $SW(W_1,W_2)$ protocol, and for the system of finite capacity coupled queues, separately even though the mathematical problem to be solved is identical for both systems. Then the main result, yielding the stationary probability distribution of the state of the protocol, as well as the joint probability distribution for the queue lengths; will be stated and proved. Measures of interest, such as the throughput of the queueing system will then be derived.

Assumptions for the $SW(W_1,W_2)$ model

We shall suppose that the time necessary to transmit a packet from N1 to N2 is exponentially distributed of average value λ_1^{-1} ; the corresponding quantity in the reverse direction is λ_2^{-1}. The transmission time is measured from the instant at which the sender begins its transmission to the instant at which the receiver has received the whole packet and thus includes the transmission delay (if any). During the transmission time only one packet travels in any given direction. Furthermore, the time necessary for the exchange of aknowledgements (in state (W_1,W_2)) will also be exponentially distributed of average value α^{-1}. This last assumption is not really necessary : the results we will prove hold even though this time has an arbitrary distribution with average α^{-1}.

Assumptions for the coupled queue model

The arrivals to Q1 constitue a Poisson stream of rate λ_1; customer arrivals to Q2 are Poisson of rate λ_2. When the queue length of Q1 attains W_1, arrivals are lost

or rejected ; equivalently we may assume that the arrivals are stopped at that time. The same assumption is made when the length of Q2 attains W_2. When both Q1 and Q2 attain their maximum capacity, a service epoch of average duration α^{-1} takes place ; all customers are served (batch service) and both queues become empty at the end of the service epoch. No arrivals occur during the service time. Though we shall assume that this service time is exponentially distributed, the results will hold in the case of an arbitrary distribution.

The transition equations for the stationary probability distributions and their solution

We denote by $p(i,j)$ the probability that the protocol is in (i,j) in steady state, or that the length of Q1 is i and Q2 is j. The $p(i,j)$, $0\leq i\leq W_1$, $0\leq j\leq W_2$, satisfy the following equations :

$$p(0,0)[\lambda_1+\lambda_2] = \alpha\, p(W_1,W_2) \tag{1}$$

For $1\leq i<W_1$, $1\leq j<W_2$:

$$p(i,j)[\lambda_1+\lambda_2] = \lambda_1\, p(i-1,j) + \lambda_2\, p(i,j-1) \tag{2}$$

For $0\leq i<W_1$:

$$\lambda_1 p(i,W_2) = \lambda_1\, p(i-1,W_2) + \lambda_2\, p(i,W_2-1) \tag{3}$$

For $0\leq j<W_2$:

$$\lambda_2 p(W_1,j) = \lambda_1\, p(W_1-1,j) + \lambda_2\, p(W_1,j-1) \tag{4}$$

with the usual convention that a term $p(x,y)$ is zero if x or y is negative. Finally

$$\alpha p(W_1,W_2) = \lambda_1 p(W_1-1,W_2) + \lambda_2 p(W_1,W_2-1) \tag{5}$$

and the normalizing condition is

$$\sum_{i=0}^{W_1}\sum_{j=0}^{W_2} p(i,j) = 1 \tag{6}$$

Theorem 1. The unique solution to the system (1)-(6), yielding the stationary probability distribution for the $SW(W_1,W_2)$ protocol states, and to the coupled queue problem, is given by :

For $0\leq i<W_1$, $0\leq j<W_2$:

$$p(i,j) = \binom{i+j}{i} x_1^i x_2^j\, p(0,0) \tag{7}$$

For $0\leq i<W_1$:

$$p(i,W_2) = \frac{\lambda_2}{\lambda_2} x_2^{W_2-1} \sum_{k=0}^{i} \binom{W_2+k-1}{k} x_1^k\, p(0,0) \tag{8}$$

For $0\leq j<W_2$:

$$p(W_1,j) = \frac{\lambda_1}{\lambda_2} x_1^{W_1-1} \sum_{k=0}^{j} \binom{W_1+k-1}{k} x_2^k\, p(0,0) \tag{9}$$

$$p(W_1,W_2) = \frac{\lambda_1+\lambda_2}{\alpha}\, p(0,0) \tag{10}$$

where $x_1 = \lambda_1/(\lambda_1+\lambda_2)$, $x_2 = \lambda_2/(\lambda_1+\lambda_2)$, and for $W = \min(W_1,W_2)$

$$(11)\qquad p(0,0) = \Big[1 + \frac{\lambda_1+\lambda_2}{\alpha} + \sum_{k=1}^{W_1} k\, x_1^{W_1-k} \binom{W_2-1+W_1-k}{W_1-k} \frac{\lambda_2}{\lambda_1}^{W_2-1} x_2 + \sum_{k=1}^{W_2} k\, x_2^{W_2-k} \binom{W_1-1+W_2-k}{W_2-k} \frac{\lambda_1}{\lambda_2}^{W_1-1} x_1 + W-1 + \sum_{\substack{W\le i<W_1 \\ W\le j<W_2}} \binom{i+j}{i} x_1^i x_2^j \Big]^{-1}$$

Proof : To verify (7) to (10), it suffices to show that they satisfy (1) to (5). That (10) satisfies (1) is obvious, and it is easy to see that (7) satisfies (2). Let us show that (8) satisfies (3) : it suffices to show that $p(i,W_2-1)$, which is given by (7), is equal to

$$(12)\qquad \frac{\lambda_1}{\lambda_2}\cdot\frac{\lambda_2}{\lambda_1}^{i} x_1^{i} x_2^{W_2-1} \binom{W_2+i-1}{i} p(0,0)$$

But from (7) we have

$$(13)\qquad p(i,W_2-1) = \binom{W_2+i-1}{i} x_1^i x_2^{W_2-1} p(0,0)$$

which establishes the result.

Remark. The model for which we have provided the solution is not a special of the known finite capacity queueing models. Thus the solution does not have the usual "product form" [3].

THROUGHPUT OF THE $SW(W_1,W_2)$ PROTOCOL

The maximum throughput achievable by the protocol in each direction can be computed directly from the above result. Let T_1 and T_2 denote the throughput, in packets transmitted per unit time, from N1 to N2 and from N2 to N1 respectively.

Notice that N1 transmits packets in all states (i,j) of the protocol, except those with $i=W_1$. Since a packet requires λ_1^{-1} units of time to be transmitted from N1 to N2, we obtain

$$(14)\qquad T_1 = \lambda_1\Big[1 - \sum_{j=0}^{W_2} p(W_1,j)\Big]$$

and similarly

$$(15)\qquad T_2 = \lambda_2\Big[1 - \sum_{i=0}^{W_1} p(i,W_2)\Big]$$

These expressions for throughput are a useful measure allowing us to compare this protocol to others, in particular to HDLC.

First consider the case where $\lambda_1 = \lambda_2$ (symmetric traffic) and $W_1 = W_2 = W$. Assume that the length of the acknowledgement packet is negligible compared to that of information packets ($\lambda \ll \alpha$) ; throughput in any one direction versus W is given in

Table 1. On the same table we give the throughput of HDLC, derived in [5], under the same conditions.

Table 1

W	1	2	3	4
SW(W,W)	0.667	0.728	0.763	0.785
HDLC	0.5	0.75	0.875	0.9375

Throughput versus window width for SW(W,W) and HDLC under symmetric traffic.

We see that HDLC is clearly superior even for small values of window size under symmetric traffic and when the length of ACKs is very small.

When the ACK bits in an HDLC packet are an important part of each packet the trade-off between $SW(W_1,W_2)$ and HDLC becomes more complex to evaluate. For $SW(W_1,W_2)$, the ACK traffic in any one direction is limited to $\lambda/W_i\alpha$ of the throughput while for HDLC it will be W_i times larger. On the other hand, the effective waiting time for an ACK will be greater with $SW(W_1,W_2)$, since with HDLC an ACK is transmitted with each packet.

For large values of W one can argue that SW(W,W) will be superior because the relative importance of the waiting time for an ACK will become negligible. The argument is as follows. Let A be the total time for transmitting W packets from N1 to N2, and let B be the corresponding quantity in the reverse direction. As W becomes large, by the central limit theorem A will become normally distributed of mean W/λ and coefficient of variation tending to zero ; a similar statement can be made about B. Therefore, of the system is under symmetric traffic, the probability that $A > B$, or that $B > A$ will tend to zero since A and B will both tend towards constants of the same value. Thus the throughput θ in either direction will have the form

$$T \cong \lambda/(1+\frac{\lambda}{W\alpha}) \tag{16}$$

The corresponding quantity for HDLC is [5]

$$T_{HDLC} \cong \lambda(1-\frac{1}{2^W})/(1+\frac{\lambda}{\alpha}) \tag{17}$$

and we see that for large W :

$$T/T_{HDLC} \cong W(\frac{\alpha+\lambda}{W\alpha+\lambda}) \tag{18}$$

for symmetric traffic. Thus we can expect that SW(W,W) will be superior to HDLC for large values of window size whenever the ACK packets (or bits) cannot be neglected.

In the case of non-symmetric traffic and different window widths W_1,W_2, a similar analysis can be carried out. By virtue of the central limit theorem, the time necessary for N_1 to transmit W_1 packets will be approximately gaussian of expected value W_1/λ_1; if σ_1^2 is the variance of packet length, the variance of this time will be approximately $W_1\sigma_1^2$ so that its coefficient of variation will be $\lambda_1\sigma_1/\sqrt{W_1}$. The same analysis being valid for N_2, we see that if (say) $W_1/\lambda_1 > W_2/\lambda_2$, we obtain

$$T_1 \cong \lambda_1 \frac{W_1/\lambda_1}{\frac{W_1}{\lambda_1}+\frac{1}{\alpha}} = \lambda_1/(1+\lambda_1/\alpha W_1) \tag{19}$$

and

$$T_2 \cong \lambda_2 \frac{W_2/\lambda_2}{\frac{W_2}{\lambda_2}+\frac{W_1}{\lambda_1}-\frac{W_2}{\lambda_2}+\frac{1}{\alpha}} = \lambda_2/\left(\frac{\lambda_2 W_1}{\lambda_1 W_2}+\frac{\lambda_2}{\alpha W_2}\right) \tag{20}$$

because N_2 will remain idle for, approximately, a time of length $(W_1/\lambda_1)-(W_2/\lambda_2)$ after having transmitted its W_2 packets.

For HDLC we shall have [5] :

$$T_1' \cong \lambda_1/\left(1+\frac{\lambda_1}{\alpha}\right) \tag{21}$$

$$T_2' \cong \lambda_2/\left(1+\frac{\lambda_2}{\alpha}\right) \tag{22}$$

so that

$$T_1/T_1' \cong W_1\left(\frac{\alpha+\lambda_1}{W_1\alpha+\lambda_1}\right) \tag{23}$$

and

$$T_2/T_2' \cong \left(\frac{\alpha+\lambda_2}{\alpha\frac{\lambda_2 W_1}{\lambda_1 W_2}+\frac{\lambda_2}{W_2}}\right) \tag{24}$$

Let us define

$$k = \frac{W_1}{\lambda_1} \Big/ \frac{W_2}{\lambda_2}$$

By assumption, and without loss of generality $k>1$. Then, by equation (24) we can see that $T_2>T_2'$ whenever

$$\alpha(k-1) < \lambda_2\left(\frac{W_2-1}{W_2}\right)$$

or whenever the (approximate) inequality

$$\frac{k-1}{\lambda_2} < \frac{1}{\alpha} \tag{25}$$

holds. Thus we see that for large W_1, W_2 and asymmetric traffic, $SW(W_1,W_2)$ yields a higher throughput from N_1 to N_2 in all cases (see equation (23)), and also a higher throughput from N_2 to N_1 whenever (25) holds.

Acknowledgments

The authors are grateful to Mrs M.Th. BOUVIER for her efficient preparation of the manuscript.

References

[1] A. Dantine, E. Eschenauer (1976) : Influence on packet node behavior of the internode protocol. IEEE Trans. on Communications, vol. 24, n° 6, 606-614.

[2] G. Fayolle, E. Gelenbe, G. Pujolle (1978) : An analytic evaluation of the performance of the "send and wait" protocol. IEEE Trans. on Communications, vol. 26, n° 3, 313-319.

[3] E. Gelenbe, R. Muntz (1976) : Probabilistic models of computer systems. Acta Informatica, vol. 7, n° 1, 35-60.

[4] E. Gelenbe, J.L. Grangé, P. Mussard (1977) : Performance limits of the TMM protocol ; modelling and measurement. Submitted to Computer Networks and IRIA-LABORIA Research Report 230.

[5] E. Gelenbe, J. Labetoulle, G. Pujolle (1978) : Performance evaluation of the protocol HDLC, International Conference on Computer Network Protocols, Liège, to appear in Computer Networks.
[6] International Organization for Standardization Document ISO/DIS 4335.
[7] S. Lavenberg : Throughput and maximum arrival rate of queueing systems", IBM Res. Rept. 1976, to appear in RAIRO-Informatique.

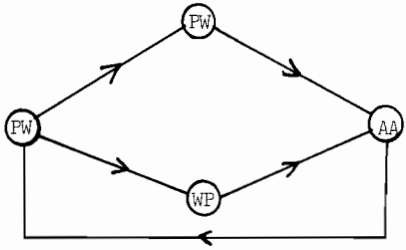


Figure 1

State transition diagram of the Send-And-Wait protocol.

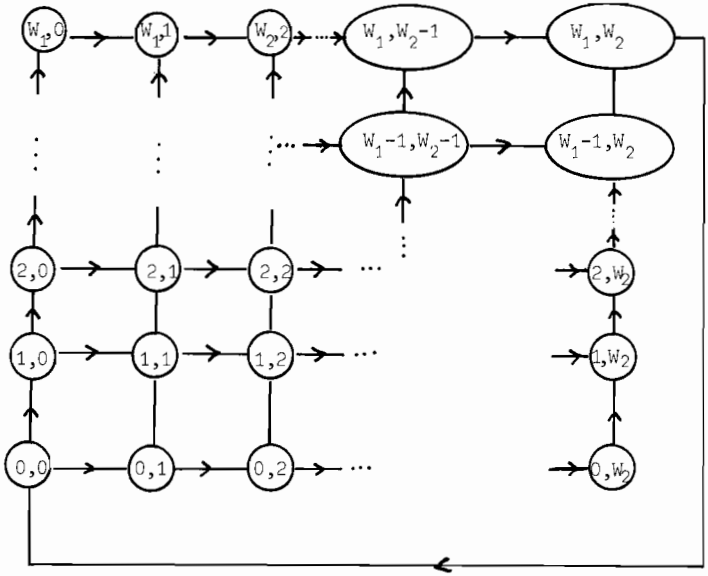


Figure 2

State transition diagram of the SW(W_1,W_2) protocol.

Performance of Computer Systems
M. Arata, A. Butrimenko, E. Gelenbe (eds.)

VIRTUAL CIRCUITS VERSUS DATAGRAM - USAGE OF COMMUNICATION RESOURCES

Alexandr Butrimenko and Ulrike Sichra

Computer Science Project
International Institute for Applied Systems Analysis
A-2361 Laxenburg
Austria

Abstract: Acceptance of X.25 international standards requires a virtual call procedure, which imposes some restriction on the usage of communication resources, namely a fixed route for the whole duration of the session. In an example of network simulation, datagram and virtual call methods are being compared, with the result that the datagram shows a better performance with respect to delivery time. The difference between both methods increases with load, number of packets per message and the size of the network.

One and probably the most significant breakthrough in computer communications has been achieved, as it is well known, due to the application of the so-called store-and-forward technique. This technique relies explicitly on the possibility of treating the pieces of a message as separate entities, which could be delivered in arbitrary order and within different time intervals. This allows a more effective usage of communication resources. In particular, increased efficiency is achieved in the following three ways:

(a) By queueing the packets at every channel through the way, instead of only having one queue at the source by circuit connection.

(b) By using the capacity of a trunk as a single high speed channel versus multiplexing it to a number of slow speed channels.

(c) By using various routes for every piece of information, depending on the actual loading of the system, and by assembling the whole message at the destination only.

The recently accepted international standard X.25 [1], based on the store-and-forward technique, requires a specific procedure, the virtual call, which provides a logical channel between source and destination for the duration of the whole session.

If the first two points (a) and (b) are applicable for any type of store-and-forwarding technique, i.e., either pure datagram service or virtual call, then the third point cannot be used for the virtual call, because the route is fixed for the whole duration of the session and will not be adapted to the changing loading situation of the network.

Table 1 represents some differences between datagram and virtual call techniques with respect to the use of communication resources. There are obviously some other significant differences that lead to the acceptance of the X.25 standard in general, but these problems are not considered here. We concentrate only on the usage of communication resources. We wish to estimate the price to be paid for a permanent route by virtual call, in terms of a possible decline in the efficient usage of communication resources.

From now on, however, we consider all packets both in the virtual call and the datagram to be equal in length, and concentrate only on differences which are caused by a fixed route for the whole session for the virtual call procedure and by free routing for every packet in the datagram procedure. A comparison of two methods has been done by simulation.

Let us describe in more detail some characteristics of the load generation process and measurements. The main performance characteristic of such a network is not the delivery of a single packet or packets, but the duration of the whole session when the complete generated message is delivered. The message generator is a Poisson process and the distribution of sources and destinations is even, according to a given matrix. As soon as a message is generated, the first packet of the message appears; the countdown of the message delivery time starts from this point.

Packets in one message are generated with exponentially distributed interarrival times and every message has the same standard number of packets to be generated. Therefore the actual load of the whole system depends on the number of simultaneously existing message generators in the system or on the number of sessions established by the virtual call simulation. Each one of these processes is finished after the required number of packets has arrived at its destination. However, it should be noted that the message delivery time is not equivalent to the duration of the corresponding generation process, because the packets require some transmission time after they have been generated. We must stress that the moment of generation of the next packet is counted from the creation moment of the previous packet. Therefore, if the interarrival intervals are shorter than the length of the packet, the packets of the same message could overlap in the time*. So the delivery time of the message will be the time period starting with the generation of the first packet of the message until all packets of the same message are delivered to their destination. For the datagram technique, a special ordering at the destination will be required, but this is not considered here. The routing technique used is based on a so-called "relief method" described earlier [2 and 3].

We do not intend to give special attention at this stage to routing problems, but only wish to emphasize that the same mechanism is used both to route datagrams and to define the route of a virtual call. As this method does not

---

*This possible source of delay, due to congestion at the origin node, will be accounted for later.

guarantee the complete absence of loops, any packet can start circulating in the network. To avoid endless loops, it is assumed that every packet has a counter, which is increased by one with every transit. As soon as this counter achieves a maximum number, which for this simulation has been set equal to eight, the packet is removed from the system. It is also assumed that the source becomes informed about the dropping of a particular packet and will re-generate it again. The time interval of the generation is the same as for the normal process of creating packets and is counted from the moment the packet has been dropped. The process of informing the source at which node the packet was removed has not been simulated.

To establish connections for the virtual call, we could use two different methods:

(a) The first packet of the message has a special calling function. Before this packet is delivered to its destination, no other packet of the message will be generated and put into the network. If the first packet is dropped, the message will merely require another attempt to establish a virtual call connection. In the simulation model, only one reason for dropping packets was considered, namely too high a number of transits. This means that as soon as the virtual call is established, no packets of that message will be dropped. Therefore, in this case, the loss of the first packet leads only to a delay in establishing a connection and in such a way as to increase the whole delivery time of a message.

(b) The second interpretation is that the generation process is not influenced by the establishing or non-establishing of a call. All subsequent packets just follow the routes of the first packet. If the first packet is dropped the generation process will be stopped and the call is considered to be lost. The rest of the packets will also be dropped - at the same point as the first one. All later results of simulating a virtual call are based on this interpretation of the virtual call setup.

Two networks (Figures 1 and 2) of different size have been simulated. Actual measurements have been made for the delivery of each packet and message. Depending on the minimum distance between source and destination, all messages (and packets) have been divided into classes. For example, in Figure 2, a message to be delivered from 1 to 4 belongs to the first class and a message to be delivered from 2 to 12 belongs to the second class, independent of the actual number of t transits that it will pass in the network.

The first set of simulation runs has been carried out in order to compare the delivery time of both methods with respect to the length of the messages and the number of packets per message. Table 2 shows that not only the delivery time for the virtual call is higher than the one for datagrams, but also that the delivery time for each packet of a virtual call is greater and increases with the length of the message. The load of the network was chosen in such a way that it remains constant, independent of the number of packets per call. Thus, Table 2 represents the results achieved by constant load, equal in this case to 6.6 packets per time unit in the network.

The simulation was carried out until the main measured values lay within a 0,2 interval with a probability of 95%. This accuracy has been kept for all other runs.

Similar tables have been made for a set of loads, varying between 4.3 and 10 packets per time unit with an approximate step size of one packet per time unit.

The most general results of the simulations are represented by two figures. Figure 3 shows the dependence of the delivery time for both datagram and virtual call on the load for three particular lengths of messages, 3, 5, and 9 packets each. It is evident that not only is the datagram faster but also that the difference of delay between the datagram and virtual call increases with the load.

For all simulated loadings and lengths of messages it is clear that the delivery time for a virtual call increases with its length, and the difference of delay between these two methods also increases with the length of the message.

Figure 4 depicts the delivery time measured in time units depending on the message length, for both virtual call and datagram. The results of simulations with two different loadings are shown, and one can see that the relationship is more or less linear, although the rate of increase changes with loading as well as with virtual call handling as compared to datagram handling.

These two figures demonstrate clearly that a datagram performs better in terms of the most important characteristics - delivery time, and even more, this is even true for all lengths and loadings.

Let us make a more detailed analysis of the simulation results. The delivery time of every message consists of three components:

(a) Time length of the message, i.e., number of packets and time intervals between their creation points;

(b) Distance to be passed in the network;

(c) Delay caused by queueing every packet, routing it, eventually dropping a packet, and rejecting it, plus regeneration of some packets when the network is loaded.

If we subtract from the delivery time of a message the time required to transmit it to the destination under ideal - unloaded network - conditions, we will get the delay caused by actions in point (c).

The ideal delivery time for virtual calls is characterized solely by the minimum distance between source and destination, and its length. The estimation of the delivery time is

$$del_{vc} = pack_{vc} + cl - 1 \quad ,$$

because in the chosen vc creation process the packets overlap. $pack_{vc}$ is the number of packets per message, and cl is the class (distance) to which the call belongs.

The overall ideal delivery time then depends on the ratio of possible connections and on the ratio of messages generation for the different classes.

In the datagram delivery system the ideal delay is dependent on the topology. A "rough" calculation of the ideal delivery time, biased towards the least number of outgoing channels of the net in Figure 1, has been taken to draw the simulation results of Figure 5. The number of packets per message is plotted on the x-axis, the difference between actual and ideal delivery time is shown on the y-axis. Each pair of curves is for a specific class (1 or 2) and a fixed loading.

In class 1, virtual calls suffer less delay caused by the actions of point (c), i.e., routing, queueing, etc. than datagrams, but the delivery time is still better

for datagrams than for virtual calls. In class 2, the difference of real and ideal delay is greater in virtual calls than in datagrams and increases when the net is more loaded.

As pointed out previously, the routing of the first packet of a virtual call and of all packets of a datagram is performed in the same way. Each node exchanges information with its neighbouring nodes about the situation of its queues. Therefore all nodes in the net have complete, though delayed information about the overall situation. When messages are treated as virtual calls, less information is exchanged between nodes than in the datagram treatment (see Table 3). This means that greater overload is created in the datagram connections. In our simulation, this overload has no influence on the delivery time as the "update packets" are assumed to be transmitted along channels different from those of messages.

A short analysis of some simulation runs for the net in Figure 2 shows results similar to those for the small net (see Figure 1).

In Figure 6 one can see that the delivery time of messages increases with the load and is higher in virtual calls than in datagrams. As the number of links to be passed increases, the delivery time is also raised and in class 4 (minimal distance is 4 links) the difference becomes rather large.

In a few simple examples it has been shown that a datagram is better than a virtual call and the differences increase with the load, length of message, and size of the network. This exercise was not done to prove that the virtual call is an inefficient way of communication, but rather to draw attention to the price to be paid for the advantages offered by the virtual call method.

As mentioned earlier the Poisson creation process does not ensure that the interarrival times of packets are longer than the packet length; thus packets of the same message could overlap in time at the creation node. Therefore a set of simulation runs has been carried out with a slightly changed timing of packet creation.

The argument that virtual calls behaviour is worse than that of datagrams in terms of delivery time no longer holds, because of overlapping packets. As can be seen from Figure 7 the datagram method is better than the virtual call method in terms of delivery time for different loadings of the network, and in terms of looking at different classes of packets. To actually compare the performance of the network the generation time should be deducted from the delivery time (generation time meaning average creation rate of messages + 1; in the nonoverlapping case the "idle time" is also included).

Figure 8 shows the behaviour of datagram and virtual call methods taking into account both generation procedures, the nonoverlapping and the overlapping. On the x-axis the packets created per time unit are represented. The y-axis represents the actual delay of a message in relation to its delivery time. The assumption underlying these figures is the simple idea that a message cannot arrive more quickly than the time needed to generate it. In this case the distance is not accounted for, but it still provides an upper limit for the maximal delay caused by queueing and routing in each simulation example.

TABLE 1

COMPARISON OF THE DATAGRAM AND VIRTUAL CALL IN RESPECT TO THE USAGE OF COMMUNICATION RESOURCES

	DATAGRAM	VIRTUAL CALL
Setting up of the session	none	delayed until the session is established or risk of misused resources if the first packet is lost or looped.
Channel usage due to	queueing, high speed trunks, routing for every packet	queueing, high speed channels, routing only for setting of a call
Address	full address in each packet - longer packets	full address only in the first packet - shorter packets

TABLE 2

packets/ message	PACKETS						MESSAGES					
	overall delivery		class 1 delivery		class 2 delivery		overall delivery		class 1 delivery		class 2 delivery	
	VC	DG	VC	DG	VC	DG	VC	DG	VC	DG	VC	DG
1	1.89	1.89	1.47	1.47	2.72	2.72	1.89	1.89	1.47	1.47	2.72	2.72
3	3.44	2.83	2.86	2.42	4.66	3.74	4.87	4.01	4.17	3.54	6.34	5.03
5	5.20	4.00	4.33	3.45	6.90	5.04	8.15	6.30	7.01	5.67	10.38	7.51
7	6.77	4.94	5.76	4.33	8.83	6.18	11.32	8.37	9.86	7.66	14.25	9.82
9	8.98	6.21	7.49	5.29	11.87	7.96	15.38	10.80	13.34	9.69	19.34	12.92
11	10.53	7.16	9.01	6.07	13.61	9.38	18.96	12.95	16.75	11.71	23.45	15.51

load: 6.6 packets/tu for net in Figure 1

TABLE 3

packets/ message	arrived update information / created packets · 100		rejected update information / created packets · 100		Σ VC	Σ DG
	VC	DG	VC	DG		
1	40.93	40.93	4.78	4.78	45.71	45.71
3	44.30	52.48	12.09	13.70	56.39	66.18
5	42.00	55.13	14.68	20.42	56.68	75.55
7	40.34	54.85	16.34	23.30	56.68	77.15
9	37.22	55.17	17.08	24.62	54.30	79.79
11	31.63	53.50	14.88	25.24	46.51	78.74

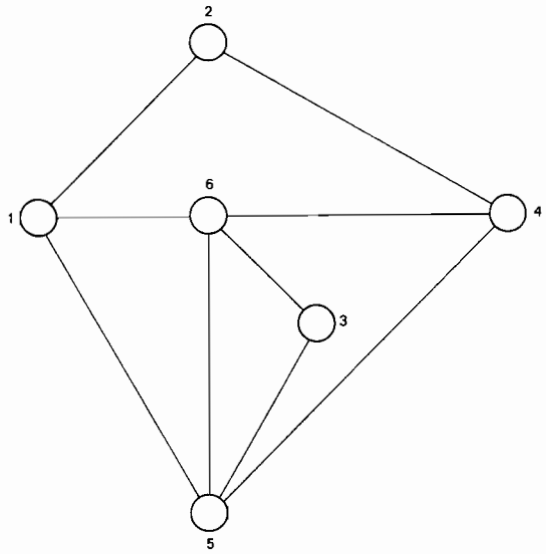


FIGURE 1

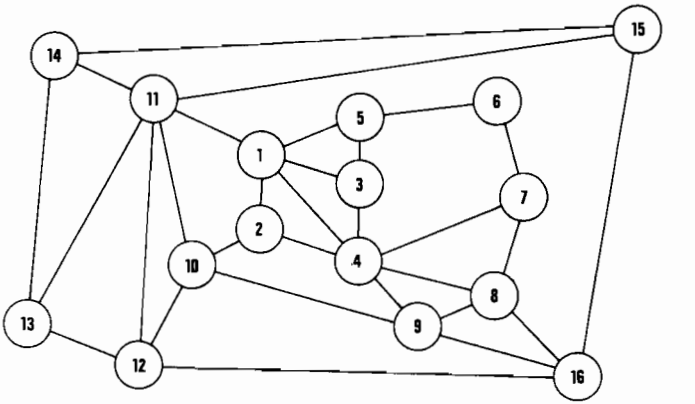


FIGURE 2

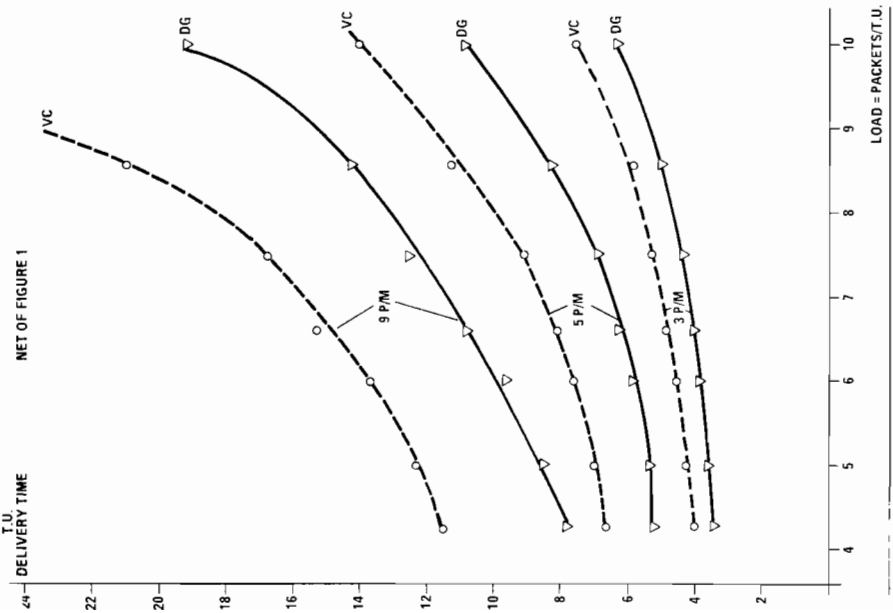


FIGURE 3

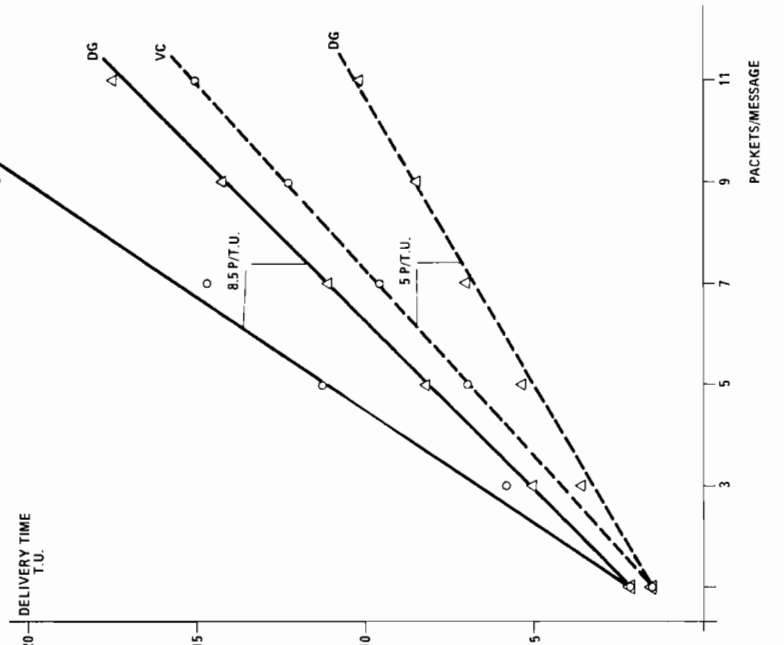


FIGURE 4

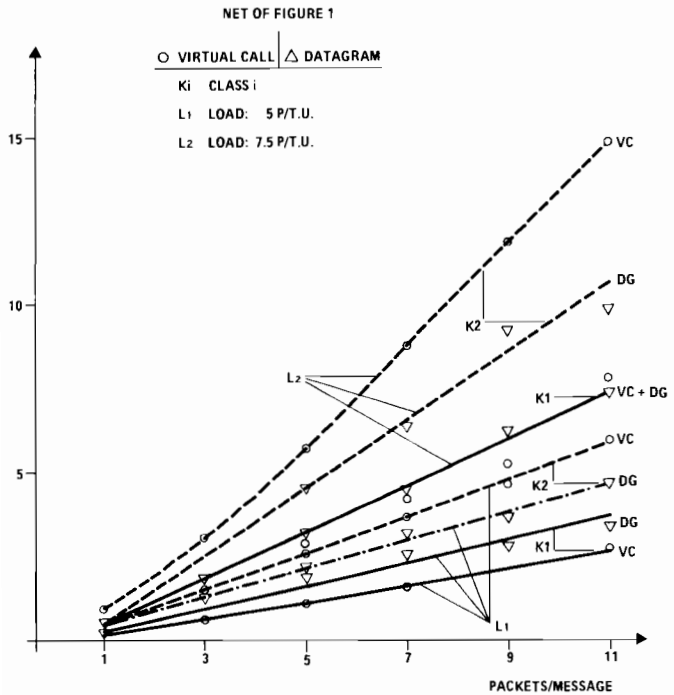


FIGURE 5

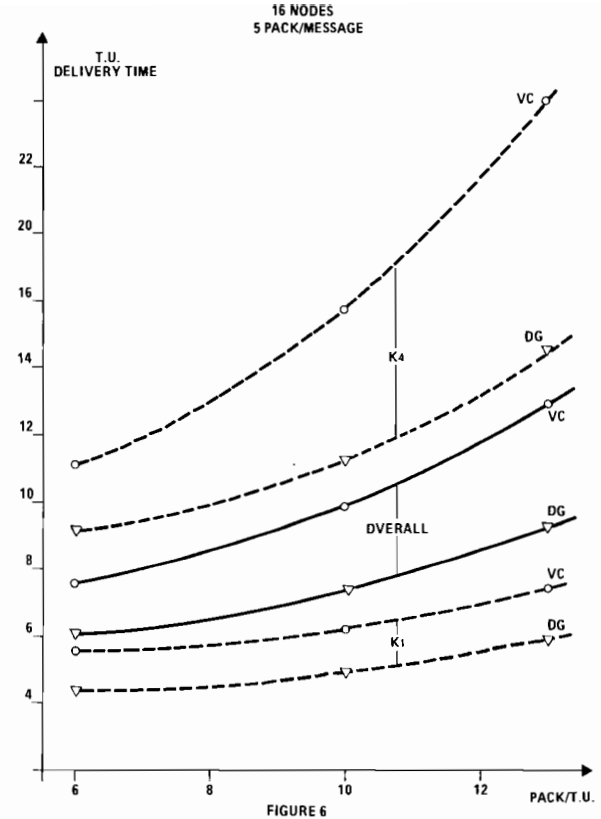


FIGURE 6

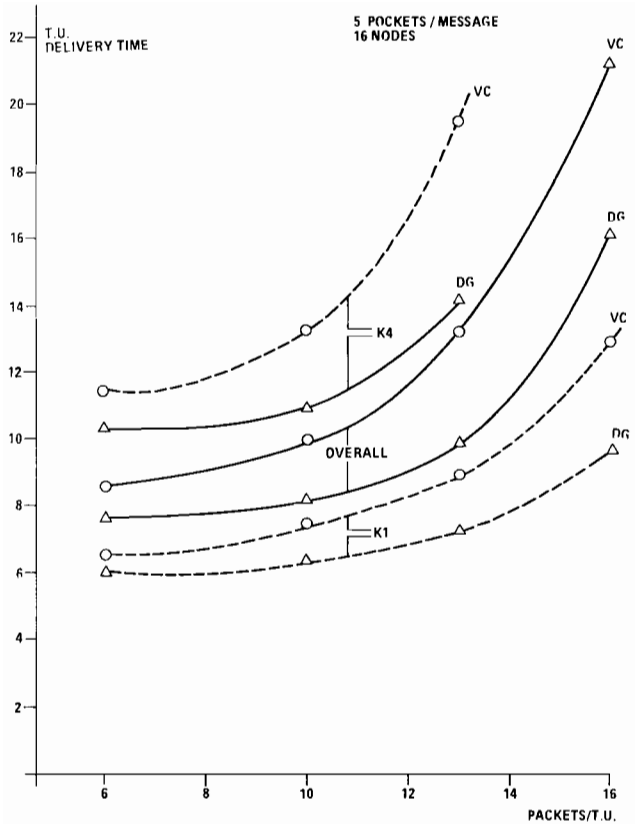


FIGURE 7

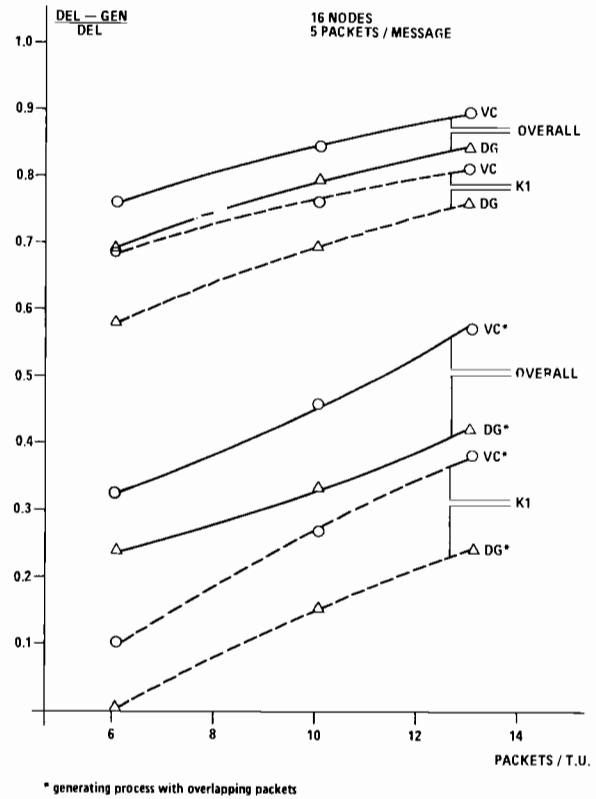


FIGURE 8

References

[1] A. Butrimenko and U. Sichra: Priorities and Pay Function Routing for Packet Switching Network, Fachtagung der GI and NTG, Aachen, BRD, March/April 1976, Springer Verlag.

[2] Interface between Data Terminal Equipment (DTE) and Data Circuit/Terminating Equipment (DCE) for Terminal Operating in the Packet Mode on Public Data Networks, X-25, in Public Data Network, Vol. VIII. 2, pp. 70-109, Geneva, 1977.

[3] Routing Techniques for Message Switching Networks with Message Outdating, Symposium on Computer Communication Networks and Teletraffic, New York, 1971.

COMMUNICATION NETWORK MODELLING II

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

FAILSAFE DISTRIBUTED LOOP - FREE ROUTING IN COMMUNICATION NETWORKS

Adrian Segall
Dept. of Electrical Engineering
Technion - Israel Institute of Technology
Haifa, Israel

We present several algorithms for routing of information in communication networks. All protocols are characterized by distributed computation, loop-freedom in the routing tables for each destination in the network, adaptability to changes in flow requirements and immunity in face of arbitrary number, location and timing of topological changes. Two algorithms converge to the minimum delay routing in message and line-switched networks respectively. A third, somewhat simpler algorithm, maintains at any given time only one route between each pair of nodes in the network and converges to the shortest paths between each source and each destination in terms of arbitrary weights on the links.

INTRODUCTION

Reliability and the ability to recover from topological changes are properties of major importance in the design of communication networks. In today's networks it happens occasionally because of technical malfunctions or, in military networks because of war actions, that nodes and links fail and then possibly recover; also new nodes or links become operational and have to be smoothly added to an already operating network. The reliability of a communication network depends on its ability to cope with these changes, meaning that no breakdown of large portions of it will be caused by such changes and that in finite - and hopefully short - time after their occurrence, the remaining portions of the network will be able to operate normally.

The present paper gives an overview of several algorithms for routing in communication networks. In order to fix ideas, we shall call them ALGORITHM A, B and C, respectively. All algorithms have the following features and properties:

a) Distirbuted computation.
b) Loop-freedom for each destination at all times.
c) Adaptability to load changes.
d) Recoverability from arbitrary number, timing and location of topological changes.

In addition, each algorithm brings the network in steady-state to some type of optimal condition:

e) If topological changes stop after a certain instant for long enough time and during this time the requirements are stationary, ALGORITHMS A and B bring the network to minimal average delay routing for message and (virtual) line-switched disciplines, respectively. If the links are assigned arbitrary

---

This work was supported by the Advanced Research Projects Agency of the U.S. Department of Defense, under Contract N00014-75-C-1183.

weights that are time-invariant for long enough time, and during this time topological changes do not occur, then the third algorithm (ALGORITHM C) provides in steady-state the shortest route between all node pairs in the network.

In the following sections we shall describe the algorithms and discuss in some detail their features. Algorithms A and B for networks with no topological changes were introduced in |1|,|3|, respectively; in the same references, proofs of their properties b), c), e) are given. The extension of these algorithms to take into account topological changes and complete proofs of recoverability appear in |6|,|5|. Proofs of the properties of Algorithm C are given in |4|,|2|. In Sections 2 and 3 we shall describe the algorithms for networks without topological changes and in Section 4 we introduce their extension to cover the possibility of such changes.

UPDATING AND REROUTING

Loop-freedom of routing is a desirable property in communication networks, because it saves network resources and message delays. As said in the Introduction, our algorithms indeed maintain loop-free routing tables at all times for each destination, but in our case, this property has an additional important purpose. Loop-freedom defines a per-destination partial ordering of the nodes in the network, which is used by the protocols for the propagation of updating information and of rerouting commands. To understand this propagation, it will be useful to look at a specific network example and at several possible partial orderings for a given destination (Figs. 1a,b). Also, since the loop-freedom property and the corresponding partial ordering hold on a per-destination basis, and since the algorithms work independently from destination to destination, it will be useful to look from now on at the algorithm for a given destination, which we denote by SINK. Observe that in Fig. 1a each node has one and only one path to the destination, and therefore the partial ordering defines a tree, whereas in Fig. 1b nodes may have several paths to the SINK. This is the main feature distinguishing ALGORITHM C from ALGORITHMS A and B, and will be discussed in more detail later. We may mention here that ALGORITHM C, which maintains only one route to the destination, does not provide the policy for actual splitting of the traffic. For example, in a line-switched network, this route can be used to establish new connections, while the already established connections are not rerouted. On the other hand, ALGORITHMS A and B provide policies for actual routing of information, so that minimum average delay is reached in steady state. As such the latter give better performance, but ALGORITHM C is much simpler.

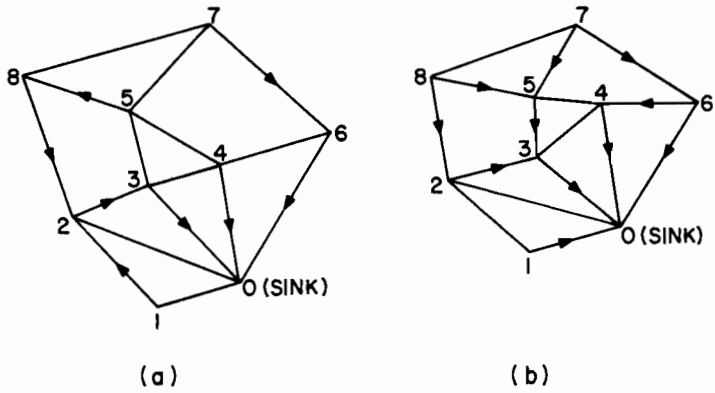


Fig. 1 - Examples of networks and of routing partial orderings.

As a matter of definition, if node k is on a path from node i to the SINK, we say that k is downstream from i and that node i is upstream from node k. Since the routing is loop-free, no node can be both upstream and downstream from any other node (for a given destination). Also, if node k is the next node after i on a path from i to the SINK, we say that k is a son of i and i is a father of k. For example, in Fig. 1b nodes 5,6 are sons of 7, and nodes 7,8 are fathers of 5.

As said before, the partial ordering defined by the loop-free routing sustains the propagation of updating control messages and of the timing of rerouting. An update cycle is started by the SINK by sending control messages to all its neighbors. A cycle consists of, first, propagation of update control messages over the partial ordering from the SINK upstream and second, propagation of rerouting commands downstream towards the SINK. The interesting fact here is that this procedure allows the SINK to be informed of the termination of a cycle, so that no new cycle will be started before the previous one has been completed. In addition, if the link delays for transmitting control messages are finite and no topological changes occur, then each cycle will be completed in finite time (for proofs, see |3|,|2|).

Next, we describe the node algorithms allowing the two waves of a cycle to propagate. As said before, a cycle is started by the SINK when it sends control messages to all its neighbors and during a cycle exactly one control message is sent over each link in each direction (for each destination).

A node can be in one of two states: normally it is in state S1, whenever it receives control messages from all its sons, it goes to state S2, and by the time it receives control messages from all neighbors the node returns to state S1. In addition to the list of its sons, each node i also maintains an estimated distance d_i to the destination where this distance is measured in terms of some link weights that are possibly time-varying, but always strictly positive. In the transition from state S1 to S2, a node, i say, takes the following actions: decide upon the new d_i using the control messages corresponding to the current cycle received up to now and send a control message of format (SINK, d_i) to all neighbors except the sons. The exact policy for calculating the new d_i is different from algorithm to algorithm and is not very important at this stage. One can see that the transition from S1 to S2 is exactly the updating part of a cycle and propagates from the SINK upstream. Next consider the second part of a cycle, namely the propagation downstream from the network towards the SINK. A node i will perform the transition from S2 back to S1 only after receiving control messages from all its neighbors. During this transition, node i takes the actions of first sending a control message (of format (SINK,d_i)) to all its sons and then rerouting, while possibly choosing new sons and cancelling old ones. Again, the exact strategy of rerouting is not important at this stage, except that one needs to be careful not to close loops when choosing new sons. Strategies to insure this appear in Section 3.

Returning to the propagation of signals in the network, one can easily see that with the above algorithm no node can perform the transition from S2 to S1 before all its fathers do. Therefore, the first to perform this transition will be nodes without fathers, and the return to S1 will propagate downstream. The last node to return to S1 will be the SINK, and this provides signal to the SINK that the cycle has been completed, namely all nodes have returned to S1. The SINK can start a new cycle at any time afterwards.

LOOP-FREEDOM

Observe that nodes can form loops only when choosing a new son. It is at this stage that care must be used to prevent such a loop from forming.

Consider first ALGORITHM C, where only one route is maintained from each node to

the destination. One of the main theorems regarding this algorithm (see |2|) states:

Theorem 1

At all times the pattern defined by the son - father relationship is a tree rooted at SINK with the following properties:

a) if one defines S2 > S1, then the states are nondecreasing when moving downstream.

b) for all nodes in S1, the estimated distance d_i to the SINK, is strictly increasing.

The full proof of the Theorem appears in |2|, but roughly speaking, it proceeds by induction as follows: Suppose the properties hold up to time t^-. Since a node, i say, can change his son only during the transition from S2 to S1, it is easy to see that it will not close a loop if we insure that the newly chosen son, k say, has distance d_k strictly less than d_i. This is because all nodes upstream from i are already in S1 (from part a) of the Theorem) and they all have distances strictly larger than d_i, which implies that k cannot be upstream from i. The policy for calculating d_i and choosing the new son assures that $d_k < d_i$. Whenever node i received a control message (SINK,d) from neighbor ℓ, it calculates $d + d_{i\ell}$, where $d_{i\ell}$ is the weight of the link (i,ℓ) and stores this in memory as $D_i(\ell)$. The estimated distance d_i is calculated at the time of transition from S1 to S2 as the minimum of all $D_i(\ell)$ over all neighbors ℓ from which i has already received control messages during the current cycle. On the other hand, the new son k is chosen by i at the time of transition from S2 to S1 as the neighbor with minimum $D_i(\ell)$, where the minimum is taken over all neighbors. With this policy, it is clear that $d_k < d_i$.

The procedure to prevent loops in ALGORITHMS A and B is somewhat more complicated. This is because these algorithms maintain more than one route to the destination, and more than that, because reaching optimum delay requires a certain step-size for the amount of rerouting allowed during each cycle. As such, it is possible in these algorithms that a node, i say, will have a son, k say, with estimated distance $d_k > d_i$. In this situation loops are prevented as follows: if a node i has a son k with $d_k > d_i$ and node i is not sure that it can cancel all traffic through k during the present cycle, then node i declares itself blocked for the present cycle at the time of transition from S1 to S2. This blocking information is sent as an extra-flag together with all control messages of the present cycle. Any node hearing that a son of his is blocked, declares itself blocked too and sends an extra-flag attached to all control messages. In this way, node i and all nodes upstream from it become blocked for the present cycle. The protocol then requires that no blocked node can become a new son of any node during the present cycle. The full proof that blocking prevents loops for ALGORITHMS A and B appears in |1|,|3|.

TOPOLOGICAL CHANGES

For networks with topological changes it is necessary to use counter numbers m for the updating-rerouting cycles. It is possible that when topological changes occur, a previously started cycle cannot be completed, and if the SINK decides to start a new cycle before completion of the previous one, it will increase the

cycle number m. If a cycle has been completed, then the SINK may use the same cycle counter number for the next cycle. Exact procedures to trigger the start of a new cycle by the SINK even if the previous one has not been completed (because of a topological change) will be described presently. Nodes will also have counter numbers n_i corresponding to the cycle counter number, and all control messages will now have the format (SINK,m,d) where - if node i is the sender - then $m = n_i$ and $d = d_i$. The counter number n_{SINK} is exactly the counter number of the last cycle started by the SINK. Also, a node i receiving a message (SINK,m,d) from a neighbor k, stores m in a cell $N_i(k)$ in addition to storing $d + d_{ik}$ in $D_i(k)$.

For ALGORITHM C, a failure in the network may happen on a tree-link or on links that are not on the tree (failures of nodes are considered as failures of all links adjacent to the node, so that they need no special attention). In either case, the SINK must be informed (by a distributed protocol to be presently indicated) not to wait for the completion of the present cycle, because this completion may never occur, and to start a new cycle with a higher counter number. In addition, if the failure occurs on a tree-link, all nodes upstream of the failure must be informed that their route to the destination has been disconnected and therefore not to wait for updating control messages over the tree. For example in Fig. 1a, if link (2,3) fails, then nodes 1,5,8 lose their route to the destination.

We next describe the propagation of failure information upstream and the actions taken by the nodes receiving such information. A node whose link to its son fails, enters a state S3 and sends a message (SINK,m,$d = \infty$) to all its remaining neighbors. A node receiving such a message from its son, also enters state S3 and sends a similar message to all neighbors except its son. In this way, all nodes upstream from a tree-link failure eventually enter state S3. A node i entering S3 looks at its tables $N_i(k)$ for messages with counter numbers higher than its own node counter number, and if none are present, waits for such messages to come. In the former case, it picks the neighbor with smallest $D_i(k)$ as the new son, in the latter, it picks the neighbor from which it first receives such a message as the new son. At the time this happens, the node transits to state S2, chooses the corresponding $D_i(k)$ as its new estimated distance d_i to the SINK, sends control message (SINK,m,d) to all neighbors except the new son, and from now on proceeds as usual.

Bringing a link up is done by a similar protocol. If a link between two nodes i and k becomes operational, the two nodes compare their counter numbers via a local protocol and choose the maximum of those as the number to use for bringing the link up. Node i actually brings link (i,k) up whenever it receives a message with counter number strictly higher than this maximum, either from its best link or from k. A similar algorithm holds for k.

Finally, we describe the (distributed) protocol for informing the SINK that because of a topological change, the present cycle might not be completed, so that the SINK will know to immediately start a new cycle with a higher counter number. A node, i say, discovering a failure or a link (i,k) becoming operational sends a special control message REQ(m) to its son (if i is not in S3), where $m = n_i$ in the case of failure, or m equals $\max(n_i, n_k)$ for links becoming operational. A node ℓ receiving a message REQ(m) forwards it to its son if ℓ is not in S3 or destroys it if ℓ is in S3. Clearly, a REQ-message may encounter another failure or a node in S3, namely with no routing path to the SINK, in which case it gets destroyed. However, one of the main theorems of |2| states that if a message REQ(m) is generated, then some message REQ(m) will positively arrive at the SINK in finite time. The protocol requires that if the SINK receives REQ(m), then it will immediately start a new cycle with counter number (m+1), provided such a cycle was not started before (because of a REQ(m) message previ-

ously received). It is proved in |2| that with this protocol the network will recover in finite time, in the sense that all nodes physically connected to the SINK will have a routing path to it.

The extension of ALGORITHMS A and B to take into account topological changes is similar in principle to the one for ALGORITHM C, except that one has to separately consider situations when a node has only one or more than one son. The full procedure is described in |5|.

References

|1| R.G. Gallager: A minimum delay routing algorithm using distributed computation, IEEE Trans. on Comm., COM-25 (1977) 73-85.
|2| P.M. Merlin and A. Segall: A failsafe distributed routing protocol, submitted to IEEE Trans. on Comm.
|3| A. Segall: Optimal distributed routing for line-switched data networks, IEEE Trans. on Comm., COM-27 (1979), January; to appear.
|4| A Segall, P.M. Merlin and R.G. Gallager: A recoverable protocol for loop-free distributed routing, Proc. of Intern. Conf. on Comm., Toronto, Canada, 1978.
|5| A. Segall and M. Sidi: Optimal failsafe distributed routing in data-communication networks, in preparation.
|6| M. Sidi: M.Sc. Thesis, Dept. of Electrical Engineering, Technion, Haifa, Israel, Feb. 1979.

Performance of Computer Systems
M. Arato, A. Butrimenko, E. Gelenbe (eds.)

SIZING A MESSAGE STORE SUBJECT TO BLOCKING CRITERIA

E. Arthurs and J. S. Kaufman
Bell Telephone Laboratories

A common theme in computer applications is that of a number of users contending for a shared resource. One example of a shared resource is storage (primary or secondary), which often must be shared by a large number of messages having varying size and residency requirements. In this paper we analyze the blocking that messages with different storage requirements incur, in contending for a common message store. Examples are presented which indicate the effect that various traffic parameters have on blocking and which also illustrate trade-offs inherent in a message storage environment. These results are applicable to sizing message stores in a variety of applications such as store and forward networks, call answering systems and disk storage systems.

INTRODUCTION

In recent years, considerable attention has focused on computing the blocking that different data streams incur, in contending for a common multiplexed data channel [1-8]. The basic queuing model studied in these papers (from a data multiplexing viewpoint) turns out to be appropriate in studying the blocking that different size messages incur, in contending for a common message store. This latter problem arises naturally in sizing message stores subject to blocking criteria [9]. Message stores can be found in an increasing number of applications such as store and forward networks, call answering systems and disk storage systems.

In this paper, in addition to our new application of the above mentioned model, we present new results on both the theoretical and computational aspects of the queuing model. These results include

i) Replacing the previously assumed exponential service time distributions [1-8] by general service time distributions.

ii) Eliminating a problem encountered by previous authors in dealing with a huge state space - which effectively prevented numerical computations [7,8]. This problem was solved by developing a Buzen type recursion [10]. All quantities of interest are formulated and efficiently computed in terms of a recursively computed partition function.

iii) Developing a number of easily computable and quite accurate approximations, which are appropriate for back-of-the-envelope type calculations. One of these approximations uses a very effective but relatively little known "peaked traffic" characterization [11,12].

The paper is organized as follows: in Section 2 we define the model and briefly discuss previous relevant work. The Markovian state equations and the known product form solution are briefly reviewed in Section 3 to set notation. The key to using this solution in problems with a large state space ($\sim 10^6$ states are common in message store problems) lies in being able to efficiently compute a normalizing constant (the partition function). In Sections 4 and 5 we show how to compute this partition function via a Buzen-like recursion and we obtain explicit expressions for various quantities of interest. Several approximations to the message blocking, which only rely on readily available Erlang B tables, are presented in Section 6. In Section 7 we study a variety of examples which illustrate the sensitivity of message blocking to various parameters. The paper concludes in Section 8 with a discussion of several useful extensions of the model.

THE MODEL

Figure 1 shows the problem under consideration in general terms. The message store has a finite capacity of c units, messages arrive at a mean rate λ and a message carries with it two requirements:

a) a spatial requirement - b units of storage

b) a temporal requirement - storage for τ units of time

The specific assumptions we make are

i) the message arrival process is a stationary Poisson process with mean rate λ.

ii) the message size b is an arbitrary discrete random variable ($P\{b=b_i\} = q_i$, $i = 1,\ldots,k$).

iii) a message with size b_i has a residency (storage) time τ which is drawn from an arbitrary distribution with mean μ_i^{-1}.

iv) the system is in equilibrium.

v) a message requiring b_i units of storage is blocked if and only if fewer than b_i units of storage is available.

vi) blocked messages depart without further effecting the system.

Although assumption v) implies complete sharing of storage by all messages, we will see that considerably greater generality is possible (e.g., portions of storage may be dedicated to specific message sizes, etc.). Also, although assumptions i) and ii) imply that messages of size b_i arrive according to a Poisson process with mean rate $\lambda_i = \lambda q_i$, the theory actually allows for finite source and/or Poisson message arrival processes.

The salient difference between the assumptions made earlier [1-8], and those made here is the generality of the storage time assumption. On the other hand, but in consonance with the classical Erlang B loss model [12,13], the detailed structure of the storage time distribution doesn't effect the equilibrium blocking phenomena - only the mean storage time $\left(\mu_i^{-1}\right)$ enters the picture. Thus,

the earlier results [1-8] are considerably more general than was realized. These comments apply only to the basic loss model studied in Reference 1-8. Several of these papers generalize the basic model in other directions, loss-delay [1], preemptive [3] and limited availability configurations [4,6]. These generalizations were studied, for the most part, via analytic approximation and simulation techniques - and the generalization to nonexponential storage time distributions does not hold for any of these generalizations.

To set notation, but also to make this paper self-contained, we briefly review the Markovian balance equations and the product form state distribution. This has been studied with varying degrees of completeness in References 1-8.

THE STATE DISTRIBUTION

If each of the c units of storage is thought of as a "server", a message of size b will require exactly b servers. Unlike blocking systems with batched Poisson arrivals [14] however, all b servers must be relinquished simultaneously. To capture this effect it is appropriate to define the system state $\underline{n}$ by $\underline{n} = (n_1,\ldots,n_k)$ where n_i = number of type i messages in storage. Clearly, the storage occupied when the system is in state $\underline{n}$ is simply $n\cdot b = \sum_{i=1}^{k} n_i b_i$. In this section we assume that the distribution of storage time $F_i(.)$ for type i messages is exponential:

$$F_i(x) = 1 - e^{-\mu_i x} \qquad x \geq 0 \tag{1}$$

and study the resulting Markovian population process. In Appendix I we briefly sketch the basic results which hold for general storage time distributions. The proofs have been omitted due to space limitations but are the subject of a forthcoming paper [15].

We begin with the case of most interest to us - namely, that where the total message store of capacity c is completely shared. Assume that the message sizes are ordered:

$$b_1 \leq b_2 \leq \cdots \leq b_k$$

and define the disjoint sets:

$$R_o = \{\underline{n}:\ o \leq \underline{n}\cdot\underline{b} \leq c-b_k\}$$

$$R_j = \{\underline{n}:\ c-b_{k-j+1} < \underline{n}\cdot\underline{b} \leq c-b_{k-j}\} \qquad j = 1,\ldots,k-1$$

$$R_k = \{\underline{n}:\ c-b_1 < \underline{n}\cdot\underline{b} \leq c\}\ .$$

Note that for all states $n \in R_j$, $j = 1,\ldots,k$ the j message sizes $b_{k-(j-1)},\ldots,b_k$ are blocked. And clearly, the set of all states $\Omega = \{n: \; o \leq \underline{n}\cdot\underline{b} \leq c\}$ is partitioned by the sets $R_o,\ldots,R_k$.

The global Markovian balance equations in Ω can be written down by inspection. Thus, for $n \in \Omega$ we have

$$\left[\sum_{i=1}^{k} \lambda_i \delta_i(\underline{n}) + \sum_{i=1}^{k} n_i \mu_i\right] P(\underline{n}) = \sum_{i=1}^{k} \lambda_i \gamma_i(\underline{n}) P(\underline{n}_i^-) \tag{2}$$

$$+ \sum_{i=1}^{k} (n_i+1)\mu_i P(\underline{n}_i^+)\delta_i(\underline{n})$$

where $\underline{n} = (n_1,\ldots,n_k)$

$$\underline{n}_i^- = (n_1,\ldots,n_{i-1},n_i-1,n_{i+1},\ldots,n_k)$$

$$\underline{n}_i^+ = (n_1,\ldots,n_{i-1},n_i+1,n_{i+1},\ldots,n_k)$$

$$\delta_i(\underline{n}) = \begin{cases} 1 & \text{if } \underline{n}_i^+ \in \Omega \\ 0 & \text{otherwise} \end{cases}$$

$$\gamma_i(\underline{n}) = \begin{cases} 1 & n_i > 0 \\ 0 & n_i = 0 \end{cases}$$

and $\lambda_i = \lambda q_i$, $i = 1,\ldots,k$.

The global balance equation (2) has the product form solution [4,6,7,8]:

$$p(n) = \prod_{i=1}^{k} \frac{a_i^{n_i}}{n_i!} \cdot G^{-1}(c,k) \tag{3}$$

where G^{-1} is a normalization constant defined by

$$G(c,k) = \sum_{n \in \Omega} \left(\prod_{i=1}^{k} \frac{a_i^{n_i}}{n_i!} \right) \tag{4}$$

and $a_i = \lambda_i \mu_i$ is the offered load of type i messages. Of course, the product form solution (3) is a consequence of the fact that local balance [16] prevails. That is, for i = 1,...,k we have:

$$a_i \gamma_i(\underline{n}) P(\underline{n}_i^-) = n_i p(\underline{n}) \quad \text{all } \underline{n} \in \Omega \tag{5}$$

If we let P_{b_i} denote the probability that a message of size i is blocked*, then

$$P_{b_i} = P\left\{ \bigcup_{j=k-(i-1)}^{k} R_j \right\} = P\{n : \underline{n} \cdot \underline{b} > c - b_i\} \tag{6}$$

where P{A} is the probability of event $A \subset \Omega$ with respect to the distribution defined by (3).

Equation 3 is a generalization of the classical Erlang B distribution [12]. The following interpretation of our model may make this relationship clearer: Consider k infinite trunk groups, with group i trunks having mean holding time μ_i^{-1}. A fraction q_i of the calls are offered to group i only, and a type i call is accepted if and only if $\sum_{i=1}^{k} n_i b_i \leq c - b_i$ where n_i is the number of calls "up" on group i. If b_i is regarded as the cost of keeping a type i call up, then a call is blocked if and only if putting it up would exceed the available budget c.

Figure 2 is a sketch of the two dimensional state space (k=2) for the case of complete sharing of storage. As mentioned earlier, the model extends to partial sharing as well. That is, the product form solution holds if the set of allowable states Ω is defined by:

$$o \leq n_i b_i \leq c_i + c_o \qquad i = 1,\ldots,k$$

and

$$\sum_{i=1}^{k} n_i b_i \leq c_o + c_1 + \ldots + c_k$$

Thus, type i messages have dedicated storage of capacity c_i, and all messages contend for (share) a common portion of storage c_o. It is clear that the normalization constant G in equation (3) depends on Ω.

We conclude this section by pointing out that the product form solution holds for a large class of storage policies which includes the complete and partial

* At an arbitrary point in time or at a type i arrival epoch - these are identical for Poisson arrival processes [12].

sharing policies already discussed as very special (but important) cases. Thus consider a storage policy to be any set of states Ω satisfying

$$\begin{aligned} &\text{i)} \quad n_i \geq 0 \quad i = 1,\ldots,k \\ &\text{ii)} \quad \text{if } \underline{n} \in \Omega \text{ and } n_i > 0 \text{ then } \underline{n}_i^- \in \Omega \text{ for } i = 1, \ldots,k \end{aligned} \tag{7}$$

Such sets were termed "coordinate convex" by Aein in [7,8] who apparently first recognized that the product form solution holds for such sets. The interpretation of such a storage policy is that a message of size b_i is accepted for storage when the system is in state $\underline{n}$ if and only if $\underline{n}_i^+ \in \Omega$.

Inspection of the global balance equation (2) will reveal that it applies, as is, to any set Ω satisfying i) and ii) above. Hence, the proof that the product form solution equation (3) applies to any coordinate convex set is immediate.

THE COMPUTATIONAL ALGORITHM

Direct use of the product form distribution eq. (3) is difficult in all but the most trivial message storage problems, because of problems associated with computing $G(c,k)$ directly. These problems stem from the cardinality of the state space, Ω, which for typical message storage problems is very large.

In this section we show how $G(c,k)$ can be computed easily and efficiently via a recursive algorithm. We limit ourselves to the case of complete sharing of storage, but the results are easily applied to the partial sharing case described earlier*. It is appropriate to comment here that the difficulties associated with computing $G(c,k)$ directly were recognized in [7,8] and the need for a Buzen like [10] recursive algorithm was discussed, but its synthesis proved elusive.

In the following development we use $[x]$ in a standard way: $[x]$ = largest integer $\leq x$. Our basic result is

lemma 1: $G(c,k)$ is computed recursively by using

$$G(j,i) = \sum_{\ell=0}^{[j/b_i]} \frac{a_i^\ell}{\ell!} G(j-\ell b_i, i-1) \qquad \begin{aligned} i &= 2,\ldots,k \\ j &= 0,1,\ldots \end{aligned} \tag{8}$$

where

$$G(j,1) = \sum_{\ell=0}^{[j/b_1]} \frac{a_1^\ell}{\ell!} \qquad j = 0,1,\ldots$$

Proof: The idea is similar to that used by Buzen [10]. Define

$$s(j,i) = \left\{ \underline{n} : n_m \geq 0,\ 0 \leq \sum_{m=1}^{i} n_m b_m \leq j \right\}$$

*Each message type has dedicated storage, and all message types share a common "overflow" store.

and

$$G(j,i) = \sum_{n \in s(j,i)} \left(\prod_{m=1}^{i} \frac{a_m^{n_m}}{n_m!} \right) \tag{9}$$

for

$$i = 1,2,\ldots,k \quad \text{and} \quad j = 0,1,\ldots,c$$

Note that n_m varies over the values $0,1,\ldots,[j/b_m]$ and focus attention on n_i. Thus we can rewrite $G(j,i)$ as:

$$G(j,i) = \sum_{\ell=0}^{[j/b_i]} \left\{ \sum_{\substack{n \in s(j,i) \\ \text{and} \\ n_i = \ell}} \left(\prod_{m=1}^{i} \frac{a_m^{n_m}}{n_m!} \right) \right\}$$

$$= \sum_{\ell=0}^{[j/b_i]} \frac{a_i^{\ell}}{\ell!} \left\{ \sum_{n \in s(j-\ell b_i,\ i-1)} \left(\prod_{m=1}^{i-1} \frac{a_m^{n_m}}{n_m!} \right) \right\}$$

$$= \sum_{\ell=0}^{[j/b_i]} \frac{a_i^{\ell}}{\ell!} G(j-\ell b_i,\ i-1)$$

which completes the proof.

Note that one can simply compute the entries of a c x k matrix, sequentially within each column, where $G(j,i)$ is the $(j,i)^{th}$ entry. The normalization constant is the last computed entry. Fortunately, all quantities of interest can be computed from this matrix without resort to the underlying distribution. In particular we have

lemma 2: The blocking that a message of size i experiences, denoted by P_{b_i}, is given by*

* Eq. (10) shows that $b_i > b_j$ implies $P_{b_i} > P_{b_j}$, since $G(x,k)$ is monotone increasing in x.

$$P_{b_i} = 1 - \frac{G(c-b_i,k)}{G(c,k)} \qquad i = 1,\ldots,k \tag{10}$$

Proof:

$$P_{b_i} = P\{\underline{n} : \underline{n}\cdot\underline{b} > c-b_i\}$$

$$= 1 - P\{\underline{n} : 0 \leq \underline{n}\cdot\underline{b} \leq c-b_i\}$$

$$= 1 - \sum_{\underline{n}\in s(c-b_i,\, k)} p(\underline{n})$$

$$= 1 - G^{-1}(c,k) \cdot \sum_{\underline{n}\in s(c-b_i,\, k)} \left(\prod_{m=1}^{k} \frac{a_m^{n_m}}{n_m!} \right)$$

$$= 1 - \frac{G(c-b_i,\, k)}{G(c,k)}$$

Thus we find that the k^{th} column of the matrix referred to above contains all the entries needed to compute the message blocking probabilities. The partition function G (.,.) also contains information about the distribution of messages in storage. As an example, we have:

Lemma 3:

$$P(n_i=x) = \frac{a_i^x}{x!} \frac{G(c-xb_i,\, k-1)}{G(c,k)} \qquad x = 0,\ldots,[c/b_i] \tag{11}$$

where the partition function in the numerator $G(c-xb_i,\, k-1)$ is computed with respect to the $k-1$ offered loads $\{a_j\}_{j\neq i}$. The proof of lemma 3 is similar to that of lemma 2 and is omitted.

We close this section by noting that the mean message store utilization

$$L = \frac{1}{C} E\{\underline{n}\cdot\underline{b}\}$$

can be obtained directly from the blocking probabilities. Thus, using Littles Law [12], we have

$$E(n_i) = \lambda_i(1-P_{b_i})\cdot\mu_i^{-1}$$

and hence

$$L = \frac{1}{c} \sum_{i=1}^{k} a_i b_i (1 - P_{b_i}). \tag{12}$$

Note that if $\mu_i = \mu$, $i = 1,\ldots,k$ and if $P_{b_i} << 1$, $i = 1,\ldots,k$ then

$$L \doteq \frac{a\bar{b}}{c} \tag{13}$$

where

$$a = \lambda/\mu$$

and

$$\bar{b} = \sum_{i=1}^{k} q_i b_i$$

is the average message size.

5. Scaling the Recursion

For sufficiently small c ($c < 10^2$), the recursion defined by Eq. (8) presents no numerical difficulties. Typical values of c, however, may easily exceed 10^6 in message storage problems. For such values of c the recursion defined by Equation (8) is useless. Thus, for example, the corresponding values of a_i are such that the weights $\frac{a_1^\ell}{\ell!}$ in the recursion may easily overflow the computer's word size. Fortunately this problem is easily addressed by appropriately scaling the recursion. Thus, define $\hat{G}(.,.)$ by:

$$\hat{G}(j,1) = \sum_{\ell=0}^{[j/b_1]} e^{-a_1} \frac{a_1^\ell}{\ell!} \qquad j = 0, 1,\ldots,c$$

and

$$\hat{G}(j,i) = \sum_{\ell=0}^{[j/b_1]} \left(e^{-a_i} \frac{a_i^\ell}{\ell!} \right) G(j-\ell b_i, i-1) \qquad i = 2,\ldots k \tag{14}$$

Comparing Equations 8 and 14, it is easy to see that

$$\hat{G}(j,i) = e^{-(a_1+\ldots+a_i)} G(j,i) \tag{15}$$

and therefore it is obvious (see Eq. 10) that

$$P_{b_i} = 1 - \frac{\hat{G}(c-b_i,k)}{\hat{G}(c,k)} \qquad i = 1,\ldots,k \tag{16}$$

The weight functions in this scaled recursion, denoted by $P_{a_i}(\cdot)$, $i = 1,\ldots,k$

$$P_{a_i}(\ell) = e^{-a_i} \frac{a_i^{\ell}}{\ell!}$$

is of course the Poisson distribution* with mean and variance equal to a_i. Therefore, for relatively small k_0, essentially all the "mass" is distributed in the interval $[\max(0,a_i-k_0\sqrt{a_i}),\ a_i + k_0\sqrt{a_i}]$. Hence the range indicated in the recursion is dramatically reduced, weight "overflow" is prevented and the error induced by this truncation is easily bounded. This scaled recursion is easily programmed and has been used to obtain the numerical results presented below in Section 7.

6. Blocking Probability Approximations

The blocking probabilities P_{b_i}, $i = 1,\ldots,k$ can be accurately and efficiently computed using the recursive algorithm described in the last section. For quick, "back-of-the-envelope" type calculations however, the approximations presented in this section have proved to be quite useful. Both approximations approximate the average blocking probability P_b, defined by

$$P_b = \sum_{i=1}^{k} q_i P_{b_i} \tag{17}$$

* Note that, as a consequence, $G(j,i)\varepsilon[0,1]$ $\quad j = 0,\ldots,c$; $\quad i = 1,\ldots,k$

and simply require access to Erlang B blocking tables which are readily available* [12]. Approximation II is typically the better of the two approximations (as examples in Section 7 will illustrate) because it incorporates message size variance as well as mean information.

Approximation I: If all message types have the same mean storage time ($\mu_i=\mu$, $i=1,\ldots,k$), consider a new model in which all message arrivals have the same size, equal to the mean message size b:

$$\overline{b} = \sum_{i=1}^{k} q_i b_i \tag{18}$$

in our original model. The average blocking seen by message arrivals in the new model is clearly

$$B(N_{eq},a),\ N_{eq} = c/\overline{b},\ a = \lambda/\mu$$

where $B(N_{eq},a)$ denotes the Erlang B blocking for a group of N_{eq}**servers and offered load a [12]. The above idea is easily generalized to arbitrary mean storage times. Thus let $r_i = \mu_i/\mu_1$ $i = 1,\ldots,k$ in what follows. We obtain:

Approx. I:
$$P_b \doteq B(N_{eq},\ a_{eq}) \tag{19}$$

where

$$N_{eq} = c/\hat{b}$$

$$\hat{b} = \sum_{i=1}^{k} \hat{q}_i b_i$$

---

* The Erlang B formula is also easily implemented on any programmable calculator.

** N_{eq} need not be integer. When Erlang B tables are used, interpolation is required [17].

$$\hat{q}_i = (q_i/r_i) \Big/ \sum_{j=1}^{k} (q_j/r_j)$$

$$a_{eq} = \sum_{j=1}^{k} (q_j/r_j) \cdot a$$

$$a = \lambda/\mu_1$$

Approximation II: This approximation characterizes the message arrival process in terms of its mean and peakedness [12] parameters, and employs Hayward's approximation [11] to estimate the average blocking probability.

Consider offering our message arrival process (k independent Poisson message arrival processes with parameters $\lambda_i = q_i\lambda$, $\mu_i = r_i\mu_1$) to an infinite message store. Let m and z denote the mean and variance-to-mean ratio (peakedness [12]) respectively of the number of units of storage occupied in this infinite store. It is easy to show that

$$m = a \cdot \sum_{i=1}^{k} (q_i/r_i) b_i \tag{20}$$

$$z = \frac{\sum_{i=1}^{k} (q_i/r_i) b_i^2}{\sum_{i=1}^{k} (q_i/r_i) b_i} \tag{21}$$

Using Hayward's approximation [11], we obtain

Approx. II: $$P_b \doteq B(N_{eq}, a_{eq})$$

$$N_{eq} = c/z$$

$$a_{eq} = m/z \tag{22}$$

7. Illustrative Examples

Each example presented in this section represents a class of examples since an example with parameters (b_i, b_2), c is identical to the class of examples (ib_i, ib_2), ic i = 1,2,...(the state space is invariant to changes in scale).

In Figures 3-7 we view the message blocking probability as a function of the size of the message store with the mean store occupancy held constant. In Figures 3 and 4 the mean message sizes are identical, but in Figure 4 the variance of the message size distribution has increased with a resulting increased spread in the message blocking probabilities. In Figure 5, the two message sizes are identical to those in Figure 4, but the message size distribution has been chosen to maximize the coefficient of variation. The mean storage times of the two message types in Figures 3-5 are identical.

In Figures 6 and 7 we hold all parameters - except for the mean storage times - identical to those in Figure 4.* In Figure 6 the smaller message has a mean storage time twice that of the larger message and in Figure 7 this is reversed.

Among the several features evident in Figures 3-7 we emphasize:

i) message blocking decreases approximately exponentially with increasing message storage size.

ii) the mean blocking P_b is approximated very well by approximation II.

iii) the ratio P_{b_2}/P_{b_1} is approximately equal to b_2/b_1.

We note in passing that if $P_{b_2}/P_{b_1} = b_2/b_1$ then $P_{b_i} = (b_i/\bar{b})\, P_b$. Hence this last observation suggests that the approximation

$$P_{b_i} \doteq (b_i/\bar{b})P_b, \quad i = 1,\ldots,k$$

(where P_b is given by expression 22) may often be useful.

In Figure 8 we hold the storage size and mean store utilization constant and vary the fraction of size b_2 messages (q_2) from 0 to 1. The solid and dashed line curves correspond to c = 160 and c = 161 respectively. At first glance, one might expect the difference in blocking levels for the two store sizes to be uniformly small. Although this is indeed the case for size b_2

---

* For each c, the ratio λ/μ_1 is identical in Figures 4, 6 and 7. Obviously, the offered loads $a_1 = \lambda/r_1\mu_1$ and $a_2 = \lambda/r_2\mu_1$ differ and consequently so does the store utilization.

messages, note the dramatic difference in blocking for the smaller size b_1 messages. This difference is due to the fact that for $c = 161$, size b_2 messages always leave at least one unit of storage available to size b_1 messages.

8. Concluding Remarks

The model discussed in this paper has been studied recently from two new and interesting points of view. In [18], the following optimization question is raised: of all coordinate convex storage policies, which policy is optimal in terms of a given cost criterion (for example, maximizing throughput)? Interesting results are proved for the 2 and 3 dimensional (k=2,3) models. In [19], the traffic overflowing the message store is studied in a manner parallel to the classical Kosten [12] analysis. Results obtained generalize the Kosten results and may find use in engineering "overflow" message stores.

References

1. L. A. Gimpelson, "Analysis of Mixtures of Wide- and Narrow-Band Traffic," IEEE Trans. on Comm. Tech., Vol. 13, No. 3, (September 1965).
2. H. Rondina, E. Aro, R. Sweet, "Traffic Analysis of Multiple Bandwidth TDM Switching Systems Utilizing Different Channel Hunting Techniques," Conference Record of 1965 IEEE International Conference on Communications, 517 - 520.
3. T. Yamaguchi, M. Akiyama, "An Integrated Hybrid Traffic Switching System Mixing Preemptive Wideband and Waitable Narrowband Calls," Electronics and Communications in Japan, Vol. 53-A, No. 5 (May 1970), 43-52.
4. O. Enomoto, H. Miyamoto, "An Analysis of Mixtures of Multiple Bandwidth Traffic in Time Division Switching Networks," Proc. 7th Int. Teletraffic Congress, Stochkholm, Sweden (June 1973), 635/1 - 635/7.
5. G. F. W. Fredrikson, "Analysis of Channel Utilization in Traffic Concentrators," IEEE Trans. On Comm., Vol. Com-22, No. 8 (August, 1974), 1122 - 1129.
6. L. Katzschnor, R. Scheller, "Probability of Loss of Data Traffics with Different Bit Rates Hunting One Common PCM-Channel, Proc. 8th Int. Teletraffic Congress. Melbourne, Australia (November, 1976), 525/1 - 525/8.
7. J. M. Aein, G. S. Kosevych, "Satellite Capacity Allocation," Proc. of the IEEE, Vol. 65, No. 3 (March 1977), 332 - 342.
8. J. M. Aein, "A Multi-User-Class, Blocked-Calls-Cleared Demand Access Model," IEEE Trans. on Comm., Vol. Com-26, No. 3 (March 1978).
9. J. S. Kaufman, "Sizing a Message Store Subject to Blocking Criteria," Unpublished Bell Laboratories Memorandum (October 1977).
10. J. P. Buzen, "Computational Algorithms for Closed Queuing Networks with Exponential Servers," Comm. of the ACM, Vol. 16, No. 9 (September 1973), 527 - 531.

11. A. A. Fredericks, "A Development of Hayward's Approximation - Regions of Applicability," Unpublished Bell Laboratories Memorandum (August, 1975).
12. R. B. Cooper, "Introduction to Queuing Theory," the MacMillian Company, New York (1972).
13. L. Takács, "On Erlang's Formula," The Annuals of Math. Stat., (1969), Vol. 40, No. 1, 71 - 78.
14. L. Kosten, "Stochastic Theory of Service Systems," Pergamon Press Ltd., (1973).
15. E. Arthurs, J. S. Kaufman, "A Generalized Sevastyanov Theorem," in preparation.
16. L. Kleinrock, "Queuing Systems," Vol. 2, John Wiley and Sons, (1976).
17. D. L. Jagerman, "Some Properties of the Erlang Loss Function," B.S.T.J., Vol. 53, No. 3 (March 1974), 525 - 551.
18. G. J. Foschini, B. Gopinath, J. F. Hayes, "Optimum Sharing of Processors," Bell Labs Memorandum in preparation.
19. J. S. Kaufman, "A Generalized Kosten Analysis of the Message Storage Model," Bell Labs Memorandum in preparation.

Appendix I

Basic Results Obtained for General Storage Time Distributions

Let I_k be the set of k dimensional vectors with nonnegative integer components, and let Ω be any coordinate convex subset of I_k. We denote the state of the message store at time t by $\underline{N}(t)$ and assume

i) same as page 3
ii) same
iii) same
iv) $\underline{N}(0) \in \Omega$
v) a message arriving at time t and requiring b_i units of storage is blocked if and only if $\underline{N}(t) = \underline{n}$ and $\underline{n}_i^+ \notin \Omega$.
vi) same as page 3

Let $\alpha_i(t)$ denote the number of storage requests of type i arriving in (0,t] and $\beta_i(t)$ the number of blocked requests of type i in (0,t]. Our basic results are:

Theorem 1: $\lim_{t\to\infty} P(\underline{N}(t) = \underline{n}) = p(\underline{n}) \qquad \underline{n} \in \Omega$

exists with $p(\underline{n}) \geq 0$ and $\sum_{\underline{n}\in\Omega} p(\underline{n}) = 1$,

where $$p(\underline{n}) = \prod_{i=1}^{n} \frac{a_i^{n_i}}{n_i!} G^{-1}(\Omega)$$

$$G(\Omega) = \sum_{\underline{n}\in\Omega} \prod_{i=1}^{n} \frac{a_i^{n_i}}{n_i!}$$

and $a_i = \lambda_i/\mu_i$ is the offered load of type i messages.

Theorem 2: The probability of blocking of type i messages P_{b_i}, defined by

$$P_{b_i} = \lim_{t\to\infty} \frac{\beta_i(t)}{\alpha_i(t)} \qquad i = 1,\ldots,k$$

exists and is constant with probability one. It's almost certain value is

$$P_{b_i} = \sum_{\underline{n}\in V_i} P(\underline{n})$$

where $V_i = \left\{\underline{n}: \underline{n}\in\Omega \text{ and } \underline{n}_i^+ \notin \Omega\right\}$.

The proofs are omitted due to space limitations but will appear in a forthcoming paper [15].

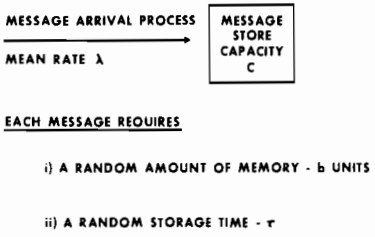


FIGURE 1 - MODEL OVERVIEW

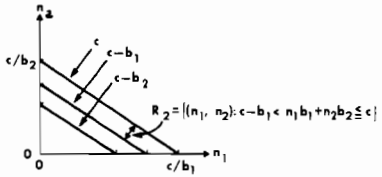


FIGURE 2 - TWO DIMENSIONAL STATE SPACE

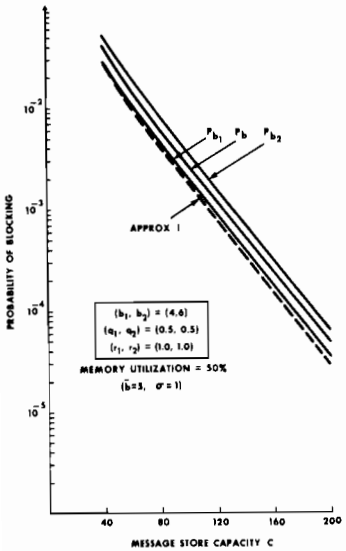


FIGURE 3

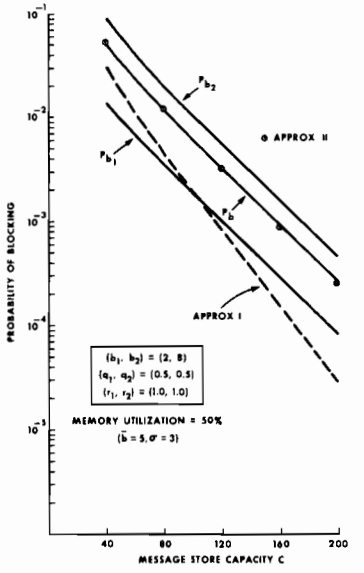


FIGURE 4

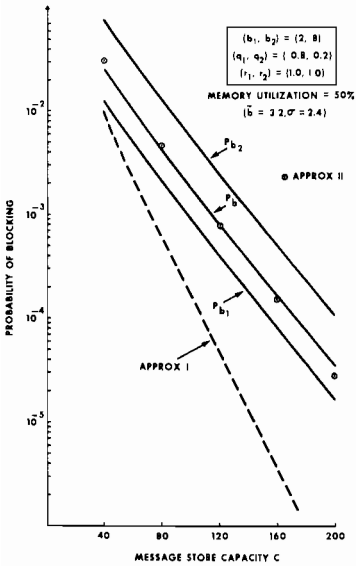


FIGURE 5

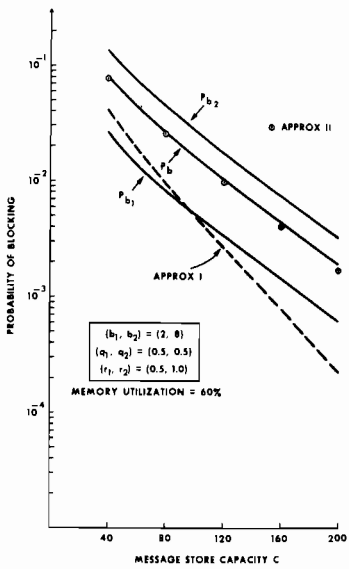


FIGURE 6

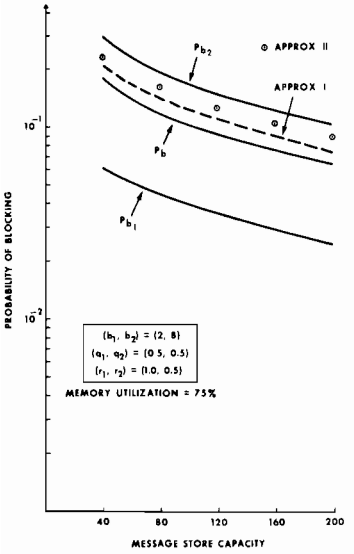


FIGURE 7

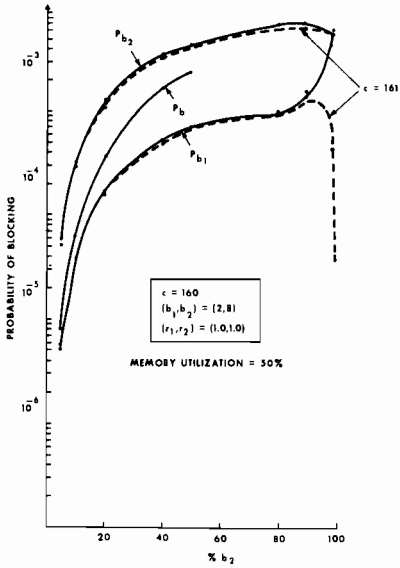


FIGURE 8

AUTHOR INDEX

ISBN 0 444 85332 4